# DOCUMENTO DE ARQUITETURA DE REFERÊNCIA - DAR

Este documento é fornecido para uso no Processo de Arquitetura de Tecnologia da Informação do Banco da Amazônia, institucionalizado pela NP 029 - Gerenciar Arquitetura de TI, no que se refere ao detalhamento, descrições e especificações técnicas dos padrões estabelecidos para: arquitetura de tecnologias para sistemas, arquitetura de tecnologias para infraestrutura, arquitetura de tecnologias para microsserviços, arquitetura de tecnologias para o Open Finance e arquitetura de tecnologias para Big Data & Analytics e Data Lake.

## HISTÓRICO DE ALTERAÇÕES

| ANO | DESCRIÇÃO | RESPONSÁVEL | VERSÃO |
|---|---|---|---|
| 2017/1 | Elaboração do Documento | Deivison Franco<br>Marcelo Portilho<br>Fábio Costa | 1.0 |
| 2017/2 | Revisão e Atualização do Documento | Deivison Franco<br>Marcelo Portilho | 2.0 |
| 2018 | Revisão e Atualização do Documento | Deivison Franco<br>Marcelo Portilho<br>Osvaldo Peixoto | 3.0 |
| 2019 | Revisão e Atualização do Documento | Deivison Franco<br>Marcelo Portilho<br>Pedro Nolasco | 4.0 |
| 2020 | Revisão e Atualização do Documento | Deivison Franco<br>Marcelo Portilho<br>Osvaldo Peixoto | 5.0 |
| 2021 | Revisão e Atualização do Documento | Deivison Franco<br>Marcelo Portilho | 6.0 |
| 2022 | Revisão e Atualização do Documento | Deivison Franco<br>Marcelo Portilho | 7.0 |
| 2023 | Revisão e Atualização do Documento | Deivison Franco<br>Marcelo Portilho | 8.0 |
| 2024/1 | Revisão e Atualização do Documento | Adriano Martins<br>Deivison Franco<br>Marcelo Portilho | 9.0 |
| 2024/2 | Revisão e Atualização do Documento | Adriano Martins<br>Deivison Franco<br>Marcelo Portilho | 10.0 |

## CONTEÚDO

**1. INTRODUÇÃO** ..... 3
**2. OBJETIVOS** ..... 3
**3. AVISO** ..... 3
**PARTE I - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE SISTEMAS** ..... 4
**1. COMPOSIÇÃO ARQUITETURAL DOS SISTEMAS** ..... 4
**PARTE II - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE INTEGRAÇÃO** ..... 5
**1. COMPOSIÇÃO ARQUITETURAL DAS INTEGRAÇÕES** ..... 5
**PARTE III - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE INFRAESTRUTURA** ..... 6
**1. COMPOSIÇÃO ARQUITETURAL DA INFRAESTRUTURA** ..... 6
1.1 Configuração Geral do Ambiente ..... 6
1.2 Rede Local e Corporativa ..... 8
1.3 Softwares de Monitoramento ..... 17
1.4 Estações de Trabalho, Notebooks e Central de Serviços ..... 17
1.5 Infraestrutura Computacional das Agências ..... 18
**2. PADRÕES MÍNIMOS PARA TECNOLOGIAS DE SERVIDORES** ..... 19
2.1 Servidores de Bancos de Dados ..... 19
2.2 Servidores de Virtualização ..... 19
2.3 Servidores de Aplicação ..... 19
2.4 Servidores Web ..... 19
2.5 Servidores de Serviço de Diretório ..... 19
2.6 Servidores de Mensageria ..... 19
**3. PADRÕES MÍNIMOS PARA TECNOLOGIAS DE REDES** ..... 20
3.1 Protocolos de Comunicação ..... 20
**PARTE IV - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE MICROSSERVIÇOS** ..... 21
**1. COMPOSIÇÃO ARQUITETURAL DOS MICROSSERVIÇOS** ..... 21
1.1 Visão e Desenho Geral ..... 21
1.2 Arquitetura do Red Hat OpenShift Container Platform ..... 21
1.3 Arquitetura de Middleware ..... 21
1.4 Arquitetura do Open Finance ..... 22
**PARTE V - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE BIG DATA, ANALYTICS E DATA LAKE** ..... 24
**1. COMPOSIÇÃO ARQUITETURAL DO BIG DATA, ANALYTICS E DATA LAKE** ..... 24
1.1 Visão e Desenho Geral ..... 24
1.2 Arquitetura do Big Data, Analytics e Data Lake ..... 25
1.3 Arquitetura das Camadas ..... 26
**ANEXO I - DOCUMENTAÇÃO DA ARQUITETURA DE SOFTWARE** ..... 30
**ANEXO II - DOCUMENTAÇÃO DO AMAZÔNIA FRAMEWORK** ..... 41
**ANEXO III - DOCUMENTAÇÃO DA INFRAESTRUTURA** ..... 50

## 1. INTRODUÇÃO

Este documento serve como referência para o processo de Arquitetura de TI do Banco da Amazônia no que se refere aos padrões tecnológicos definidos para seu ambiente computacional, na medida em que tanto o desenvolvimento (interno ou contratado de empresas terceirizadas) quanto a produção obrigam-se a produzir, adquirir implantar e implementar softwares, processos automatizados e infraestrutura que sejam plenamente compatíveis e aderentes com as tecnologias do Banco.

O padrão arquitetural aqui definido deve ser obrigatoriamente considerado e seguido quando da criação, manutenção ou aquisição de softwares e que subsidiará a elaboração de seus Planos de Arquitetura (PLAs), os quais serão avaliados pela Coordenadoria de Arquitetura de Tecnologia da Informação e Governança de Dados, da Secretaria Executiva de Tecnologia da Informação do Banco da Amazônia (SECTI/COARQ).

## 2. OBJETIVOS

Este documento visa atender a 5 objetivos inerentes aos padrões de Arquitetura de TI definidos para o ambiente computacional do Banco da Amazônia, quais sejam:

1. Consolidar informações sobre tecnologias para desenvolvimento de sistemas, recursos de hardware, software básico, ferramentas, componentes corporativos e demais configurações do Ambiente Computacional do Banco da Amazônia;

2. Descrever um modelo arquitetural genérico tanto para sistemas quanto para infraestrutura, no qual se utilizou padrões que representam uma solução adequada para o Banco;

3. Guiar tanto o desenvolvimento e a implementação de sistemas, quanto a aquisição e a implantação de infraestruturas seguindo a arquitetura definida, a fim de se obter melhor manutenibilidade de ambos;

4. Descrever como os sistemas e a infraestrutura são organizados e disponibilizados para uso;

5. Este documento não visa coibir ou restringir soluções de design que sejam diferentes das descritas aqui. O objetivo de uma arquitetura de referência não é descrever a arquitetura de um sistema específico. Isto será descrito pelo desenvolvedor e será avaliado pela SECTI/COARQ no PLA.

## 3. AVISO

É essencial que todas as implementações estejam em conformidade com os padrões e requisitos definidos e normatizados para garantir a integridade e a eficiência do ambiente de TI do Banco da Amazônia. De todo modo, caso haja a decisão de implantar soluções que sejam incompatíveis com os componentes de infraestrutura suportados ou que não estejam em conformidade com as diretrizes aqui estabelecidas, as coordenadorias de arquitetura (COARQ) e de infraestrutura (COBAN) não impedirão seu uso, mas não endossarão nem fornecerão suporte para tal. Sendo assim, a responsabilidade pela manutenção de soluções incompatíveis com a arquitetura padrão do Banco será do setor requisitante e/ou do fornecedor da solução.

Nos casos excepcionais, quando autorizados pelas instâncias superiores (Presidência, Diretoria, Secretaria e/ou Gerência), o setor requisitante e/ou o fornecedor será responsável por todo o suporte, sustentação e manutenção necessários para o pleno funcionamento da solução - incluindo requisitos como licenças de software, links de comunicação, componentes de infraestrutura, ambientes computacionais, entre outros, sem ônus para o Banco. Quaisquer necessidades ou configurações adicionais no ambiente e infraestrutura devem ser obrigatoriamente alinhadas, acordadas e validadas previamente com a Gerência Executiva de Infraestrutura e Produção (GPROD).

# PARTE I - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE SISTEMAS

## 1. COMPOSIÇÃO ARQUITETURAL DOS SISTEMAS

Toda a documentação da especificação técnica e as informações da composição arquitetural dos padrões de tecnologias estabelecidos para sistemas, bem como seus módulos, componentes, camadas, integrações, segurança, monitoramento de erros, auditoria, padrão de codificação, padrão de nomenclatura, padrão de APIs etc., além de suas representações gráficas e do detalhamento da infraestrutura para implementação de sistemas através de Processo de Integração Contínua, estão disponíveis nos endereços abaixo:

- **Documentação para Desenvolvimento de Software:**

  https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/

- **Documentação da Arquitetura de Software[1]:**

  https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/arquitetura/

- **Documentação do Amazônia Framework[2]:**

  https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/amazonia-framework/

- **Documentação da Infraestrutura[3]:**

  - **Ambiente OpenShift:**

    https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/infraestrutura/openshift/ambiente.html

  - **Ambiente de Banco de Dados:**

    https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/infraestrutura/banco-de-dados/ambiente.html

[1] A Documentação da Arquitetura de Software está contida no ANEXO I deste DAR.

[2] A Documentação do Amazônia Framework está contida no ANEXO II deste DAR.

[3] A Documentação da Infraestrutura está contida no ANEXO III deste DAR.

# PARTE II - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE INTEGRAÇÃO

## 1. COMPOSIÇÃO ARQUITETURAL DAS INTEGRAÇÕES

Cada projeto/sistema deve apresentar, em seu Plano de Arquitetura (PLA), as tecnologias de integrações utilizadas, seja com outros sistemas, seja com o ambiente de hardware/software/redes, devendo-se restringir às utilizadas pelo Banco da Amazônia, quais sejam:

- Spring Boot (https://spring.io/projects/spring-boot);
- Spring Cloud (https://spring.io/projects/spring-cloud);
- Swagger (https://swagger.io/);
- Red Hat FUSE (https://www.redhat.com/pt-br/technologies/jboss-middleware/fuse);
- Red Hat AMQ (https://www.redhat.com/pt-br/technologies/jboss-middleware/amq);
- Apache Camel (https://camel.apache.org/);
- REST APIs:
  - O padrão para desenvolvimento de REST APIs do Banco da Amazônia está disponível nos endereços:

    https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/arquitetura/padroes-api-rest.html

    https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/amazonia-framework/java/v3/modulo-rest-api.html#padrao-de-desenvolvimento-de-rest-apis
- Catálogo de APIs:
  - O Catálogo de APIs do Banco da Amazônia está disponível no endereço:

    https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/infraestrutura/ferramentas/gestao_apis.html
- Monitoramento e Observabilidade:
  - O Monitoramento e Observabilidade do Banco da Amazônia está disponível no endereço:

    https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa/pages/infraestrutura/ferramentas/observabilidade.html
- Fila de mensagens do tipo WebSphere MQ:
  - As integrações por troca de mensagens (MQSeries) deverão seguir os padrões definidos pelo Banco Central do Brasil em sua comunicação eletrônica de dados no sistema financeiro - disponível em: https://www.bcb.gov.br/estabilidadefinanceira/comunicacaodados.
- Acesso a banco de dados via db-link, bem como integrações por arquivos ou por jobs de replicação deverão ser substituídos por conexões diretas via JDBC pelo sistema.
- Não são recomendadas integrações utilizando arquivos via NFS ou SMB.

Quaisquer exceções e/ou restrições às tecnologias de integrações listadas, devem ter sua justificativa detalhada no PLA para avaliação da SECTI/COARQ.

# PARTE III - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE INFRAESTRUTURA

## 1. COMPOSIÇÃO ARQUITETURAL DA INFRAESTRUTURA

### 1.1 Configuração Geral do Ambiente

a. Software:

| | |
|---|---|
| **Sistema Operacional (ambiente "open"):** | – Windows Server. |
| **Ambiente de Banco de Dados:** | – Oracle. |
| **Servidor de Aplicação Java:** | – Apache Tomcat 2.5. |
| **Transferência de Arquivos:** | – IIS UrlScan Tool 2.0;<br>– inCD 4;<br>– Peripheral Power Controle Services;<br>– PPCS Connector. |
| **Gerenciamento de Arquivos:** | – PPCS Connector;<br>– SC Bridge;<br>– AmAPI;<br>– Symantec Backup Exec System Recovery 2010. |
| **Protocolos de Comunicação:** | – DHCP;<br>– FTP;<br>– Telnet;<br>– SSH;<br>– SMTP;<br>– TCP/IP. |
| **Escalonamento de Job's Batch:** | – SMA OPCONxps 20. |
| **Linguagens de Desenvolvimento:** | – Java;<br>– Python. |

b. Troca de Arquivos e Mensagens:

| | |
|---|---|
| **Soluções para Troca de Arquivos:** | – Connect Direct;<br>– File Transfer Protocol (FTP). |
| **Formatos para Troca de Mensagens:** | – Padrão ISO-8583 para transações financeiras dos compartilhamentos com redes externas;<br>– XML (Extensible Markup Language). |

c. Servidores Oracle Exadata:

| SÍTIO | QUANTIDADE | FABRICANTE | MODELO | CAPACIDADE (BRUTA) INSTALADA |
|---|---|---|---|---|
| Principal | 1 | ORACLE | ExaCC Quater Rack X10M (07 Nodes, 720TB) | • Distribuição de Recursos:<br>– Ambiente Produtivo: 04 Database Servers, com 400TB de armazenamento.<br>– Ambiente Não Produtivo: 03 Database Servers, com 320TB de armazenamento. |
| Secundário | 1 | ORACLE | ExaCC Quater Rack X10M (04 Nodes, 400TB) | • Distribuição de Recursos:<br>– Ambiente DR: 04 Database Servers, com 400TB de armazenamento. |

d. Servidores de Virtualização:

| QUANTIDADE | SERVIDOR | PROCESSADOR | MEMÓRIA (GB) | ARMAZENAMENTO (GB) |
|---|---|---|---|---|
| 8 | HP ProLiant DL580 G9 | 4xIntel® Xeon® E7-8860v4 | 3042 | 2400 |
| 10 | HP ProLiant DL380 G9 | 2xIntel® Xeon® E5-2637v4 | 256 | 2400 |

e. Plataforma de Software de Virtualização:

| | |
|---|---|
| **VMware:** | – vCenter Server;<br>– VMware Aria LifeCycle Manager;<br>– VMware Aria Operations;<br>– VMware Aria Operations for Logs;<br>– Site Recovery Manager + vSphere Replication. |
| **HP:** | – HPE Insight Control Manager & HPE One View. |

f. Servidores de aplicação:

| |
|---|
| – Red Hat JBoss Enterprise Application Platform;<br>– Apache Tomcat;<br>– Oracle Internet Application Server Enterprise Edition;<br>– Oracle E-Business Intelligence - Application;<br>– Oracle Business Intelligence Suite Enterprise Edition Plus;<br>– Microsoft IIS. |

g. Serviços de Mensagens:

| |
|---|
| – IBM WebSphere MQ for Linux v. 5.3;<br>– IBM WebSphere MQ for Linux v. 8.0.0.4;<br>– IBM WebSphere MQ for Linux v. 9.2.1.0. |

h. Unidades de Disco (Storages):

| SÍTIO | QUANTIDADE | FABRICANTE | MODELO | CAPACIDADE INSTALADA |
|---|---|---|---|---|
| Principal | 1 | HDS | Virtual Storage Platform | 824 TB |
| Principal | 1 | DELL | VMAX 250-F | ~120 TB |
| Secundário | 1 | HDS | Virtual Storage Platform | 490 TB |

i. Solução de Correio Eletrônico:

| | |
|---|---|
| **Servidor:** | – Microsoft Exchange. |

j. Solução de Backup Corporativo:

| SÍTIO | QUANTIDADE | FABRICANTE | MODELO | CAPACIDADE |
|---|---|---|---|---|
| Principal | 1 | Veritas | NetBackup Appliance 5230 | 112 TB |
| Principal | 1 | DELL | PowerProtect Data Domain DD9410 | 330 TB |
| Secundário | 1 | Veritas | NetBackup Appliance 5230 | 76 TB |
| Secundário | 1 | DELL | PowerProtect Data Domain DD9410 | 330 TB |

k. Serviço de Diretório (Active Directory):

| |
|---|
| – Microsoft Windows Server 2016 Datacenter;<br>– Nível funcional: Windows 2016. |

l. Sítio Interno (Intranet):

| | |
|---|---|
| **Software:** | – Microsoft Internet Explorer;<br>– Google Chrome;<br>– Mozila Firefox;<br>– Joomla;<br>– MySQL. |
| **Sistema Operacional:** | – Red Hat Enterprise Linux Server. |

m. Sítio Externo (Internet):

| | |
|---|---|
| **Software:** | – Microsoft Internet Explorer;<br>– Google Chrome;<br>– Mozila Firefox;<br>– Joomla;<br>– MySQL. |
| **Sistema Operacional:** | – Red Hat Enterprise Linux Server. |

### 1.2 Rede Local e Corporativa

a. Rede WAN:

Trata-se de uma rede privada de longa distância, que utiliza o protocolo MPLS e mecanismos de controle de qualidade de serviço QoS (Quality of Service) para atender às principais aplicações utilizadas pelo Banco.

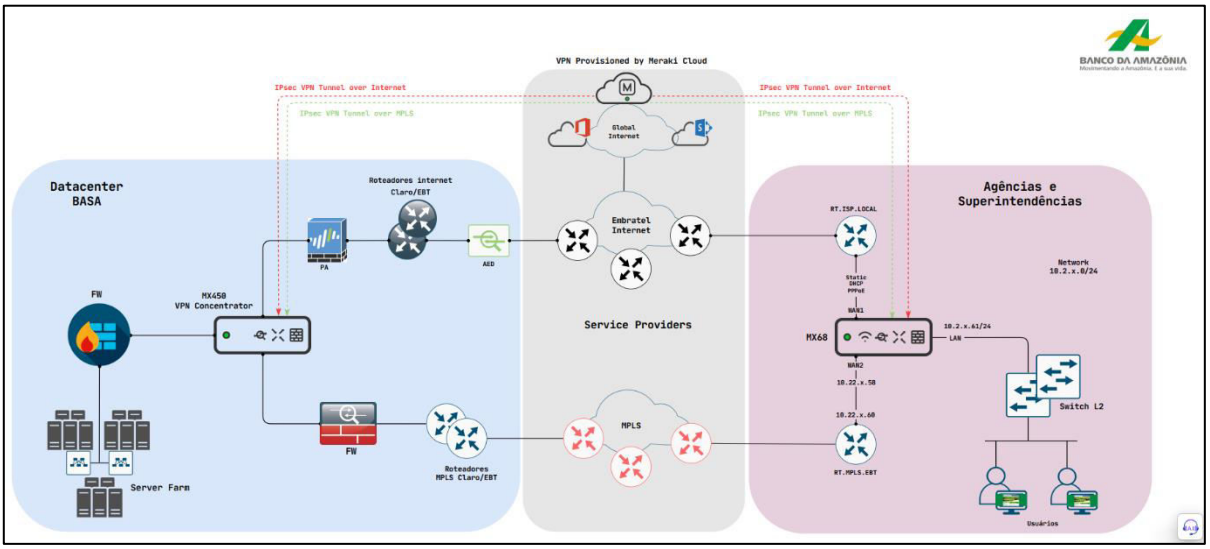
A interligação entre matriz e as agências é estabelecida por meio da rede MPLS da Embratel com link de 1G entre o concentrado MPLS na Matriz, que utiliza uma infraestrutura de fibra óptica e dois roteadores no sítio principal, além de outros dois roteadores, no sítio de processamento Secundário (site backup).

Nas agências, o acesso é garantido por um circuito principal (que pode ser fibra óptica, cabeamento metálico ou satélite) e um circuito Secundário (também em fibra óptica, metálico ou satélite), com velocidades que variam entre 10 Mbps e 300 Mbps para ambos os circuitos.

Na agência é inserido um dispositivo SD-WAN para viabilizar a comunicação segura entre a Matriz e agência através do emprego de uma solução de VPN criptografada.

As comunicações entre cliente e servidor são baseadas no padrão Ethernet com o uso do padrão TCP/IP.

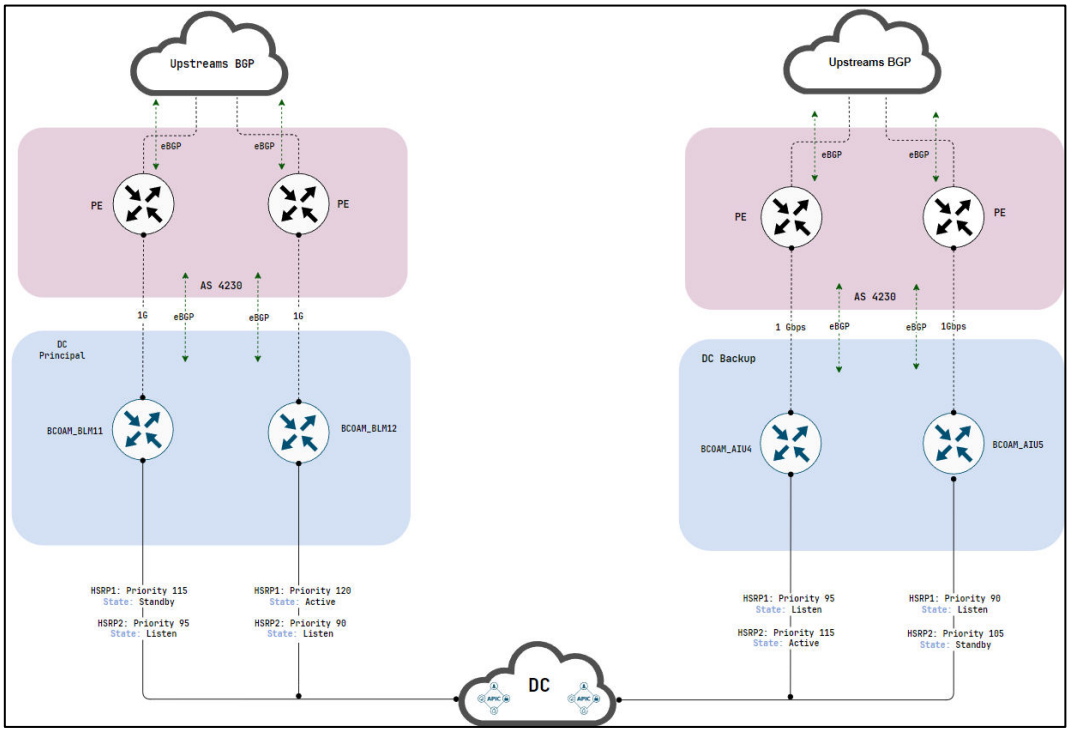
**TOPOLOGIA**




b. Rede de Acesso à Internet:

O acesso à internet é realizado através de quatro circuitos de dados fornecidos pela operadora de telecomunicações, dois no sítio principal, na velocidade de 1 Gbps e dois no sítio de processamento Secundário (site backup) na velocidade de 1 Gbps. Para aprimorar os indicadores de resiliência e alta disponibilidade é utilizado o protocolo HSRP como tecnologia de redundância de próximo salto.

**TOPOLOGIA**



c. Rede Local do Edifício Sede (Cabeada e Não Cabeada):

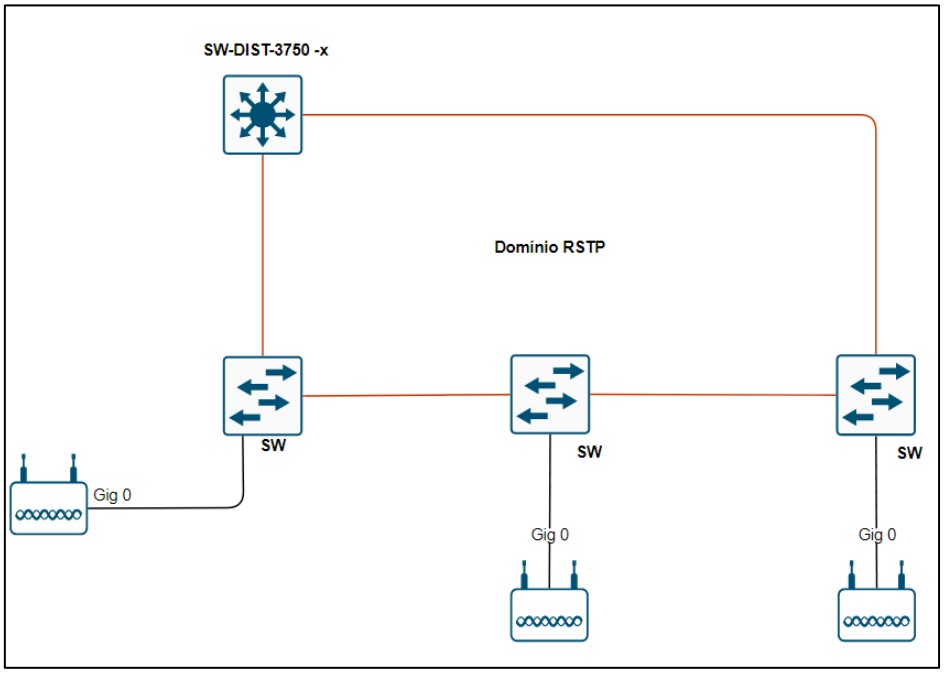
Cada andar possui de dois a seis switches para estabelecer a conectividade com os clientes finais da rede (usuários, câmeras de segurança, Pontos de Acesso Wi-Fi (AP), biometria, impressoras, telefone IP). Os switches dos andares conectam-se a um switch mais robusto que desempenha funções na camada de distribuição e utiliza a tecnologia de clustering denominada StackWise.

Em adição, os switches do edifício estão munidos com os seguintes recursos e mecanismos de segurança:

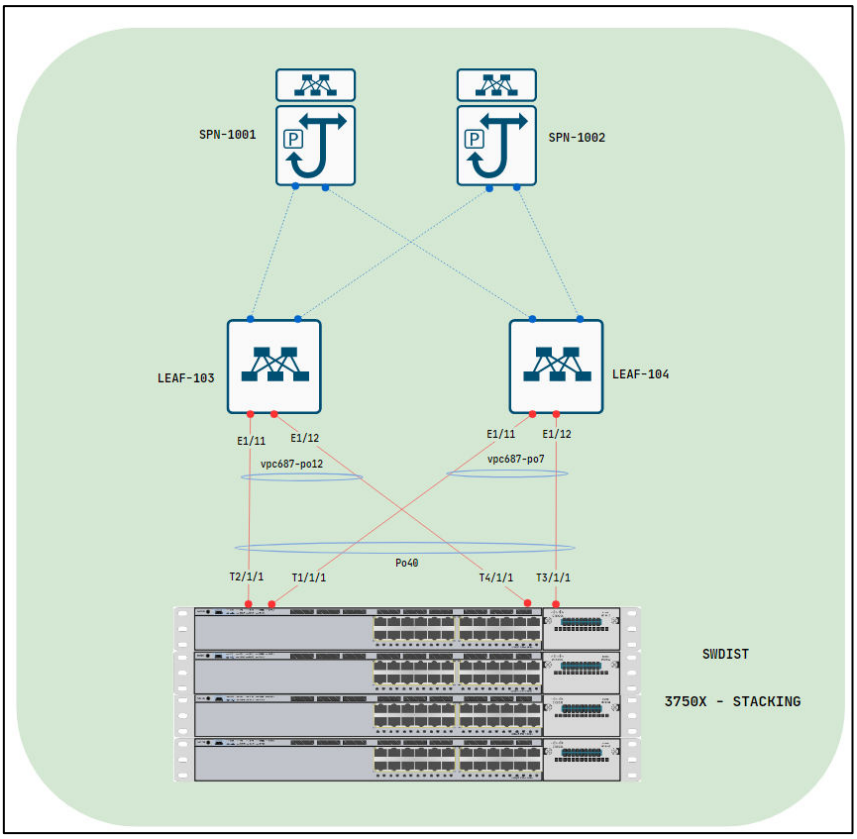
- VLAN;
- RSTP;
- 802.1x;
- ARP inspection;
- DHCP Snooping;
- Port-Security.

Toda a comunicação entre cliente e servidor é baseada no padrão Ethernet e a utilização do protocolo TCP/IP nas velocidades de 100 Mbps até 10 Gbps.

**TOPOLOGIA DOS SWITCHES DOS ANDARES**



**TOPOLOGIA DE CONEXÃO SWITCH DISTRIBUIÇÃO**




d. Datacenter

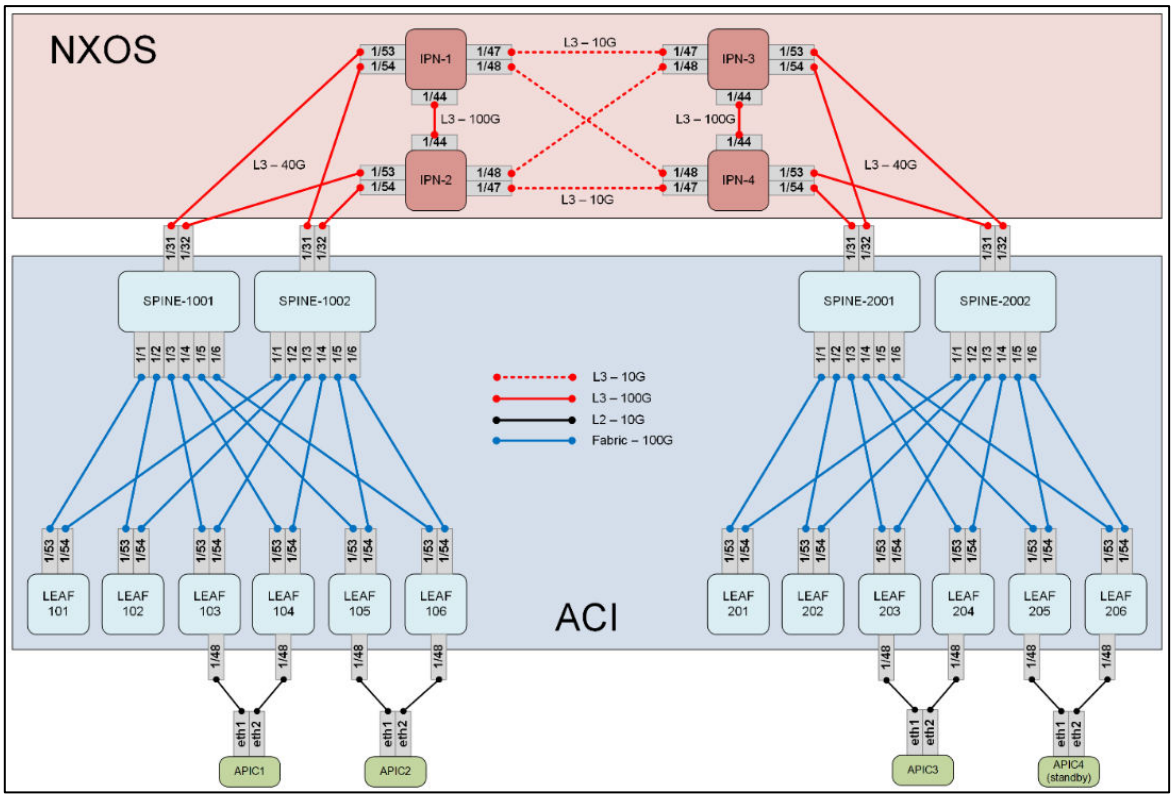
O datacenter é baseado na tecnologia de SDN (Software Defined Network) da Cisco denominada ACI (Application Centric Infrastructure). A topologia que sustenta a solução de ACI segue o modelo de arquitetura clos ou Spine&Leaf.

O Application Policy Infrastructure Controller (APIC), é o componente principal da arquitetura ACI, é composto por quatro nós, em formato de cluster ativo, é o ponto único de gerenciamento do Cisco ACI Fabric, garantia de políticas e monitoramento do ambiente.

O fabric é composto por seis switches em cada site que desempenham a função de LEAF, dois switches em cada site que desempenham a função de SPINE e dois switches em cada site que desempenham a função de IPN.

As interligações entre os equipamentos dentro do fabric operam com velocidades de 10 Gbps a 40 Gbps. A conexão entre a matriz e o site de backup é realizada por meio dos IPNs utilizando a tecnologia DWDM e na modalidade de fibra apagada. As fibras ópticas são fornecidas por operadoras de telecomunicações.

**TOPOLOGIA**




e. Segurança:

O ambiente de segurança de redes é composto por um conjunto integrado de soluções tecnológicas que garantem a proteção de ponta a ponta dos dados transmitidos na rede. Entre as principais tecnologias que fazem parte desse ecossistema, destacam-se:

**Firewalls**

Esses equipamentos utilizam uma ampla gama de recursos para proteger a rede contra ações maliciosas provenientes da internet e incidentes que possam surgir na rede interna. Os firewalls implantados no ambiente do cliente são classificados como Next-Generation Firewalls (NGFW), ou Firewalls de Próxima Geração, pois oferecem inspeção avançada e controle de acesso até a camada de aplicação.

Os firewalls estão distribuídos em quatro appliances físicos para garantir alta disponibilidade e redundância, sendo dois localizados na matriz e dois no site de backup, sendo composto por duas soluções: o Cisco Firepower e o Cisco ASA Banco, os quais estão organizados em quatro categorias, com funções específicas:

- **Firewall Interno**

  Os firewalls internos têm a função de controlar e inspecionar o acesso à rede administrativa e a outros ambientes do Banco, como os de produção, homologação, desenvolvimento, agências, parceiros, operação e SPB.

- **Firewall de Saída**

  Regula o acesso de usuários e servidores que precisam se comunicar com hosts localizados na internet.

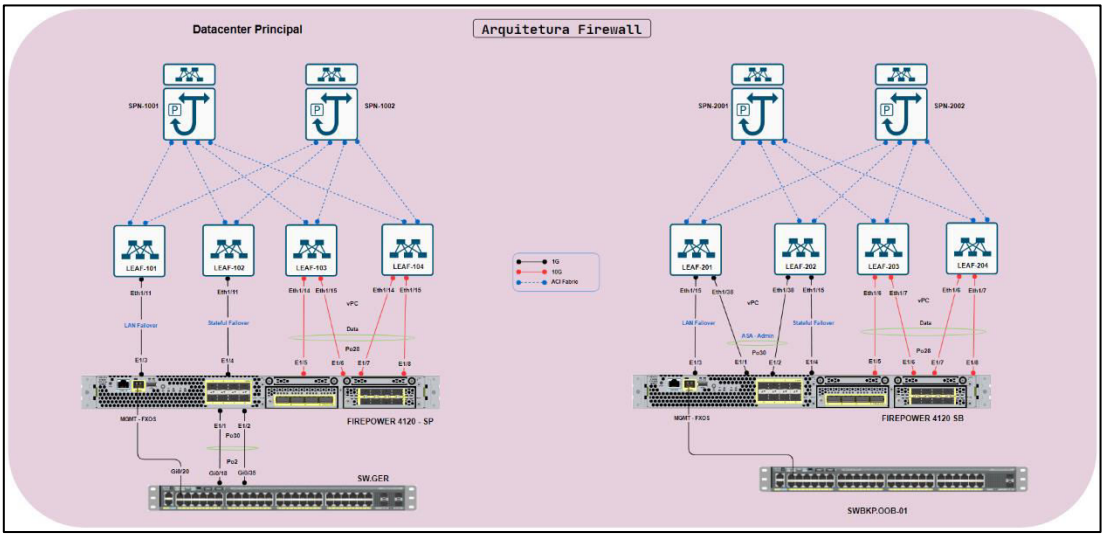
- **Firewall DMZ**

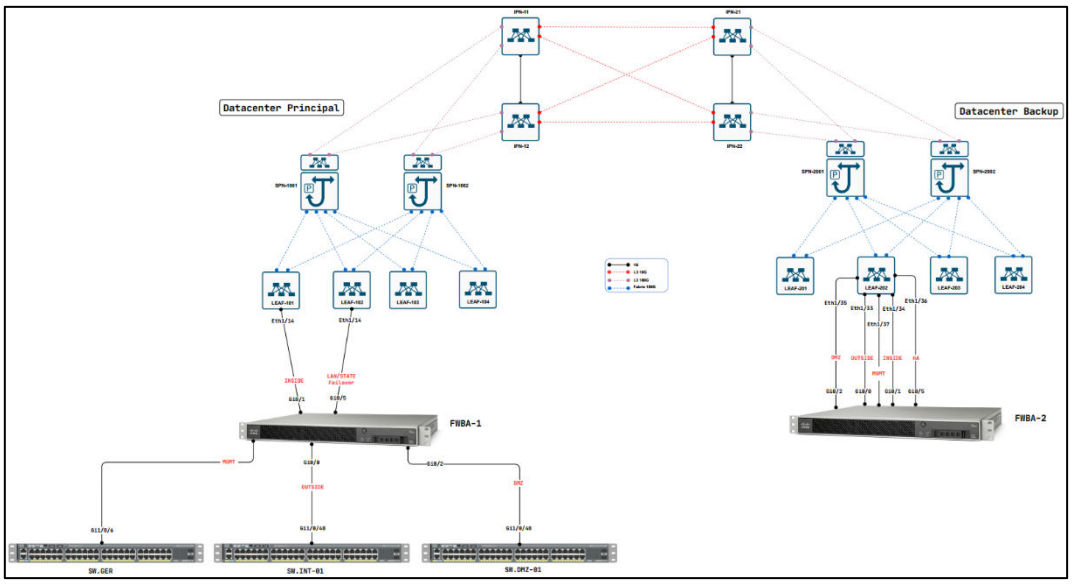
  Estes equipamentos realizam o controle de acesso para os servidores hospedados na DMZ.

- **Firewall VPN**

  Viabiliza o provisionamento de túneis VPN, tecnologia que permite o trabalho remoto e a criação de conexões seguras e privadas para comunicação com empresas parceiras do Banco.

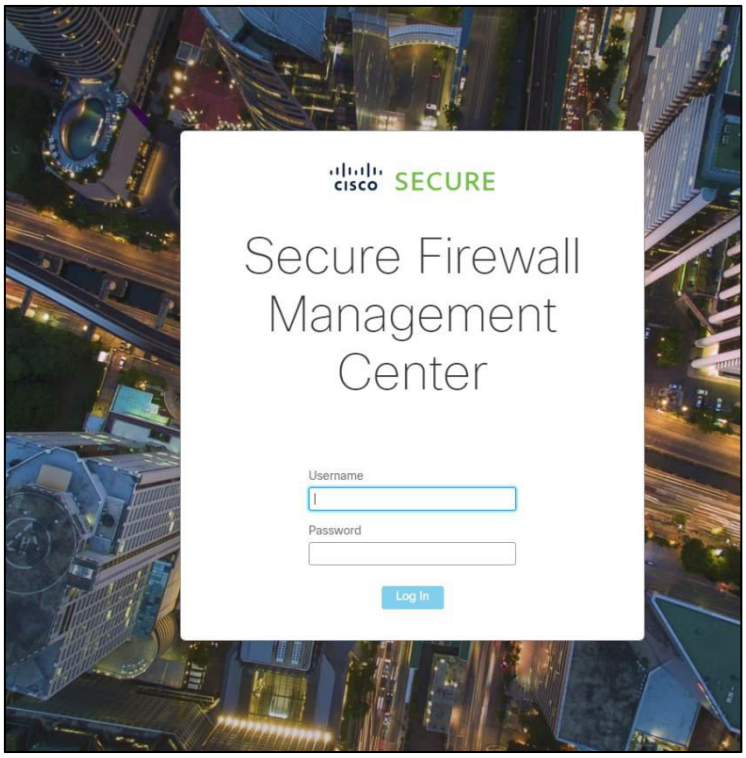
**TOPOLOGIA**



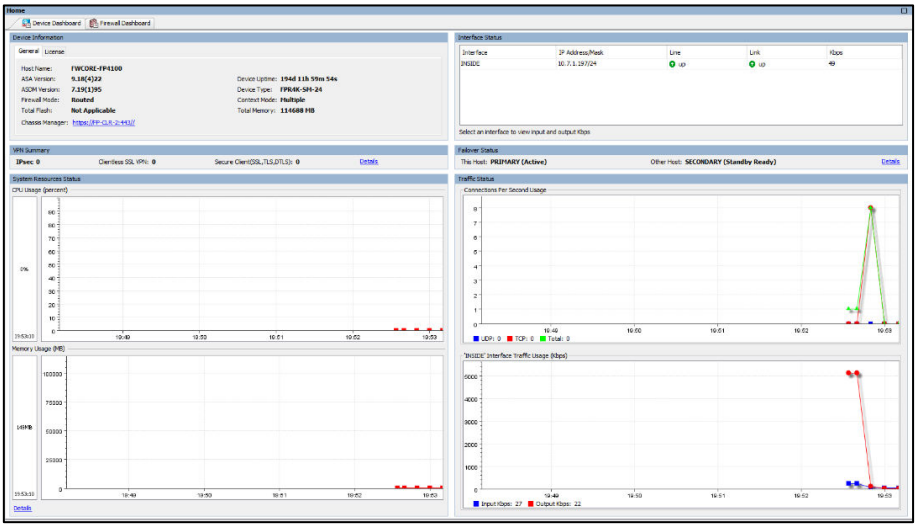




Os Firewalls são gerenciados por duas soluções distintas baseadas em interface gráfica, detalhadas a seguir.

**SFMC (Secure Firewall Management Center)**

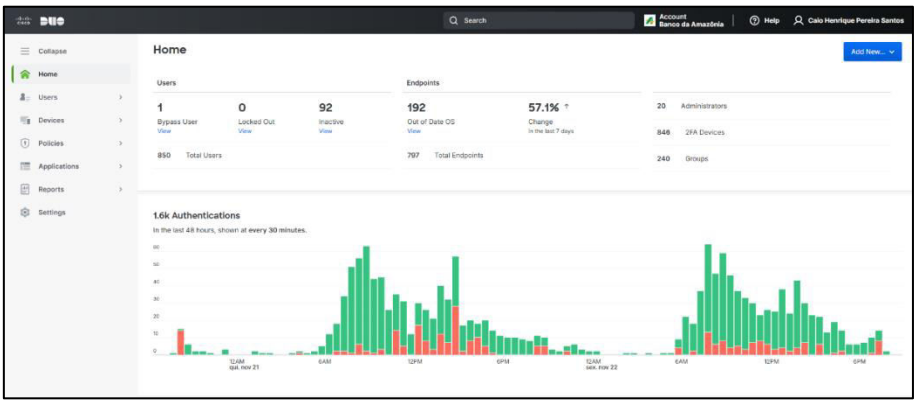



**ASDM (Adaptive Security Device Manager)**



**MFA DUO**

Trata-se de uma tecnologia de autenticação de dois fatores baseada em nuvem, desenvolvida pela Cisco. O DUO foi implantado em 2022 na matriz e agências o que aprimorou significativamente a segurança no acesso à VPN corporativa do Banco, garantindo a validação de identidade dos usuários em cada acesso.

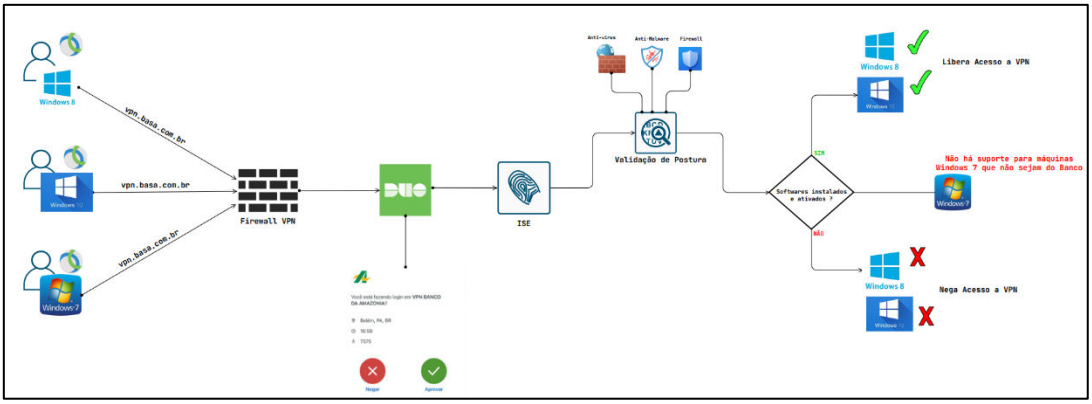



## VALIDAÇÃO DE POSTURA

É uma solução de segurança empregada para avaliar as características de um dispositivo e determinar seu nível de confiança quando este tenta conectar-se à rede corporativa do Banco através da VPN. A validação de postura ocorre antes que o processo de autenticação seja concluído. Dessa forma se o dispositivo do usuário não estiver de acordo com as políticas de segurança exigidas, a requisição de conexão à rede será negada.

Para que a postura esteja de acordo com as políticas de segurança exigidas pelo Banco, o dispositivo deve possuir os seguintes softwares instalados e ativados:

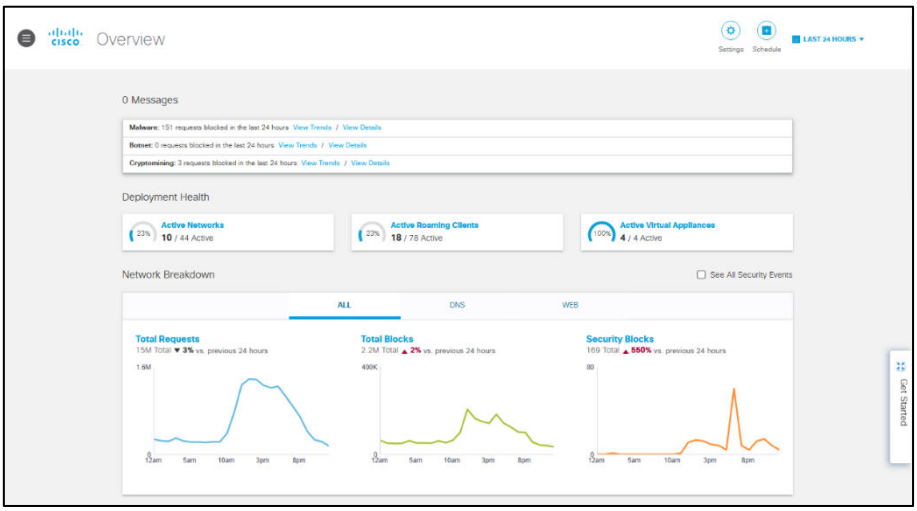
- AntiVirus - qualquer;
- AntiMalware - qualquer;
- Firewall do sistema operacional;
- DUO.




## UMBRELLA

O Cisco Umbrella é uma plataforma de segurança em nuvem que fornece uma camada adicional de defesa contra softwares maliciosos e ameaças na internet, utilizando inteligência de ameaças. Essa inteligência ajuda a prevenir o acesso a adware, malware, botnets, ataques de phishing e outros sites maliciosos conhecidos.

O Umbrella fornece proteção a nível de DNS, o primeiro invocado durante o processo de comunicação entre um cliente e um servidor. Essa característica permite que as inspeções e os controle de acesso sejam realizados antes da interação direta entre o cliente o host potencialmente malicioso.

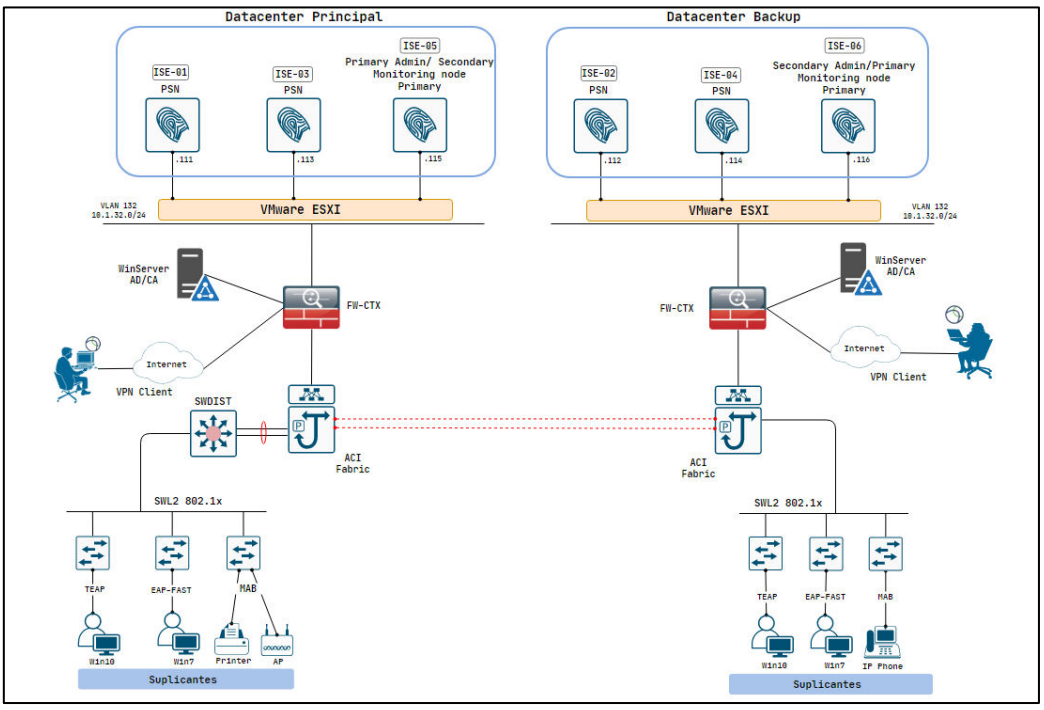



**ISE (Identity Services Engine)**

O ISE é uma de controle de identidade que simplifica a integração, o provisionamento e a proteção de todos os endpoints em uma rede, aumentando a visibilidade para identificar, classificar e monitorar com precisão todos os usuários e dispositivos conectados, permitindo que o administrador controle de maneira centralizada as políticas de acesso para os dispositivos e usuários que utilizam a rede cabeada, Wi-Fi e VPN de modo que seja possível definir quem pode acessar qual parte da rede, usando quais dispositivos e em quais circunstâncias.

Atualmente o ISE está sendo utilizado para viabilizar os seguintes serviços de segurança:

- Autenticação na VPN usando o MFA DUO e o recurso de postura;
- Autenticação através do framework 802.1x.



### 1.3 Softwares de Monitoramento

a. Monitoramento de Jobs Batch:

Para o monitoramento de jobs batch utiliza-se o OpCon/xps. que, além da automatização dos jobs, permite a automação de sistemas e fluxos de trabalhos técnicos, cuja infraestrutura está disponível em uma máquina virtual com 8 GB de RAM, um espaço de 200 GB de armazenamento e um processador de 8 núcleos.

b. Software para Monitoramento de Ativos de TI e do Datacenter:

Para esse tipo de monitoramento, utiliza-se o Zabbix e o Grafana - soluções que suportam diversas plataformas e métodos de acompanhamento, sendo altamente escaláveis para grandes ambientes, permitindo a implementação de um sistema distribuído e possibilitando uma abordagem proativa, também monitorando dispositivos de rede, recursos de sistemas operacionais, middleware, aplicações, serviços e outros elementos.

c. Software para visualização e análises gráficas

Utiliza-se o Grafana - uma plataforma que auxilia no monitoramento da disponibilidade de ativos e sistemas do Banco. Suas funcionalidades incluem a visualização e análise de métricas por meio de gráficos e fluxos, permitindo a criação de dashboards dinâmicos, que podem ser compartilhados entre equipes, contribuindo para a gestão de indicadores e o acompanhamento de dados em tempo real, proporcionando maior eficiência na tomada de decisões.

### 1.4 Estações de Trabalho, Notebooks e Central de Serviços

a. Desktops:

| | |
|---|---|
| **Sistema Operacional:** | – Windows 10 Professional 64 bits;<br>– Windows 11 Professional 64 bits;<br>– Linux Ubuntu - versões 19.04+ (apenas para as áreas de TI);<br>– Linux CentOS 6.4+ (apenas para as áreas de TI). |

b. Notebooks:

| | |
|---|---|
| **Sistema Operacional:** | – Windows 10 Professional 64 bits;<br>– Windows 11 Professional 64 bits;<br>– Linux Ubuntu - versões 19.04+ (apenas para as áreas de TI);<br>– Linux CentOS 6.4+ (apenas para as áreas de TI). |

c. Cliente Web (Navegador):

| |
|---|
| – Microsoft Internet Explorer 9+; (IE 10 será descontinuado em agosto de 2021)<br>– Microsoft Edge;<br>– Google Chrome 75+;<br>– Firefox 19+;<br>– Opera 66+. |

d. Software da Central de Serviços:

| |
|---|
| – Cherwell Service Management 4.60. |

e. Solução de Antivírus e Antispyware:

| |
|---|
| – Crowdtrike Falcon Endpoint Protection. |

## 1.5 Infraestrutura Computacional das Agências

a. Plataforma de Automação de Agências:

| | |
|---|---|
| **Rede local:** | – Rede padrão ETHERNET de 100 Mbps e 1000Mbps;<br>– Protocolo: TCP-IP;<br>– Switch 24/48 portas 100/1000 Mbps;<br>– Ponto de acesso wireless 802.11 AC<br>– Comunicação WAN com Matriz via rede MPLS (BGP).<br>– Comunicação SW-WAN com a Matriz via INTERNET. |
| **Interligação com a Rede da Direção Geral:** | – Link MPLS com protocolo TCP-IP;<br>– Link Principal de 10 Mbps a 155 Mbps;<br>– Link Secundário de 10Mbps a 200 Mbps. |

b. Terminais de Caixa:

| TIPO / MODELO / FABRICANTE | DESCRIÇÃO | SOFTWARE BÁSICO | HARDWARE |
|---|---|---|---|
| Terminal de caixa/DC5A-U/DATEN | Terminal de Caixa | Debian GNU/Linux 13 trixie/Sid<br><br>J/XFS (Java eXtension for Financial Services) | AMD Ryzen 5 PRO 4655GE com Radeon Graphics, CPU de 3.3GHz e 8GB de memória |

c. Terminais de Autoatendimento:

| TIPO / MODELO / FABRICANTE | DESCRIÇÃO | SOFTWARE BÁSICO | HARDWARE |
|---|---|---|---|
| ExtraCash/TMF4100/290.02.028/PERTO | Terminal de Autoatendimento | Debian GNU/Linux 9.11 (stretch) J/XFS (Java eXtension for Financial Services) | Intel (R) Core (TM) i3-6100 com CPU de 3.70GHz e 8GB de memória |
| ExtraCash/TMF 4100/290.01.863/PERTO | Terminal de Autoatendimento | Debian GNU/Linux 9.12 (stretch) J/XFS (Java eXtension for Financial Services) | Intel (R) Pentium (R) com CPU G2030 de 3.00GHz e 8GB de memória |
| TOTEM/TC-3100/290.02.071 /PERTO | Terminal de Autoatendimento | Debian GNU/Linux 11 (bullseye) J/XFS (Java eXtension for Financial Services) | Intel (R) Core (TM) i5-10400 com CPU de 2.90GHz e 8GB de memória |

## 2. PADRÕES MÍNIMOS PARA TECNOLOGIAS DE SERVIDORES

Os padrões a seguir são definidos como minimamente aceitáveis para os servidores que comporão a infraestrutura do ambiente computacional do Banco da Amazônia.

### 2.1 Servidores de Bancos de Dados

| | |
|---|---|
| **Hardware:** | – Exadata modalidade Cloud at Costumer Gen2<br>– CPU 56 coresMemória: 756GB;<br>– Capacidade: 299 TB. |
| **Sistema Operacional:** | – Oracle Linux x86;<br>– Oracle 19c+. |

### 2.2 Servidores de Virtualização

| | |
|---|---|
| **Hardware:** | – Processador: 4xIntel® Xeon® E7-8860v4 (2.2GHz/18-core/45MB/140W);<br>– Memória: 3 TB RAM;<br>– Capacidade: 24000 GB. |
| **Sistema Operacional:** | – VMware vCloud Foundation 7.0.3. |

### 2.3 Servidores de Aplicação

| | |
|---|---|
| **Hardware:** | – Processador: 4vCPUs;<br>– Memória: 8 GB RAM;<br>– Capacidade: 20 GB. |
| **Sistema Operacional:** | – Red Hat Enterprise Linux 8.0+. |

### 2.4 Servidores Web

| | |
|---|---|
| **Hardware:** | – Processador: 4 vCPUs;<br>– Memória: 8 GB RAM;<br>– Capacidade: 20 GB. |
| **Sistema Operacional:** | – Red Hat Enterprise Linux 8.0+. |

### 2.5 Servidores de Serviço de Diretório

| | |
|---|---|
| **Hardware:** | – Processador: 4 vCPUS;<br>– Memória: 8 GB RAM;<br>– Capacidade: 40 GB. |
| **Sistema Operacional:** | – Microsoft Windows 2016 Datacenter+. |

### 2.6 Servidores de Mensageria

| | |
|---|---|
| **Hardware:** | – Processador: 8 vCPUs;<br>– Memória: 8 GB RAM;<br>– Capacidade: 60 GB. |
| **Sistema Operacional:** | – Red Hat Enterprise Linux 8.0+. |

## 3. PADRÕES MÍNIMOS PARA TECNOLOGIAS DE REDES

Os padrões a seguir são definidos como minimamente aceitáveis para os equipamentos que comporão a infraestrutura de rede do Banco da Amazônia.

### 3.1 Protocolos de Comunicação

Os sistemas, aplicações e equipamentos necessitarão ter suporte a:

| | |
|---|---|
| **Pilha de Protocolos TCP/IP:** | **Versões:** |
| | – IPV4 e IPV6. |

| | |
|---|---|
| **Balanceamento de Carga:** | **Persistência de Sessão:** |
| | – Full. |

| | |
|---|---|
| **SMNP Community:** | **Versões:** |
| | – 2C e 3. |

| | |
|---|---|
| **Firewall:** | – NAT - Network Address Translator;<br>– PBR - Policy Based Routing. |

| | |
|---|---|
| **Roteamento:** | **Estático:** |
| | – Rotas Estáticas. |
| | **Dinâmico:** |
| | – OSPF - Open Shortest Path First;<br>– BGP - Border Gateway Protocol. |

| | |
|---|---|
| **Protocolos de Redundância:** | – VRRP - Virtual Router Redundancy Protocol;<br>– HSRP - Hot Standby Router Protocol. |

| | |
|---|---|
| **Proxy:** | – Normal Cache;<br>– Transparent. |

# PARTE IV - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE MICROSSERVIÇOS

## 1. COMPOSIÇÃO ARQUITETURAL DOS MICROSSERVIÇOS

### 1.1 Visão e Desenho Geral

A modernização tecnológica do Banco da Amazônia, descrita em seu planejamento estratégico institucional, perpassa pela implementação de produtos digitais elencados no PDTI 2024/2026. Sendo assim, se torna necessária uma nova arquitetura e barramento corporativo de serviços, que vai ao encontro da Visão do PEI 2023-2030, a qual cita a modernização tecnológica do Banco e de seus canais de relacionamento.

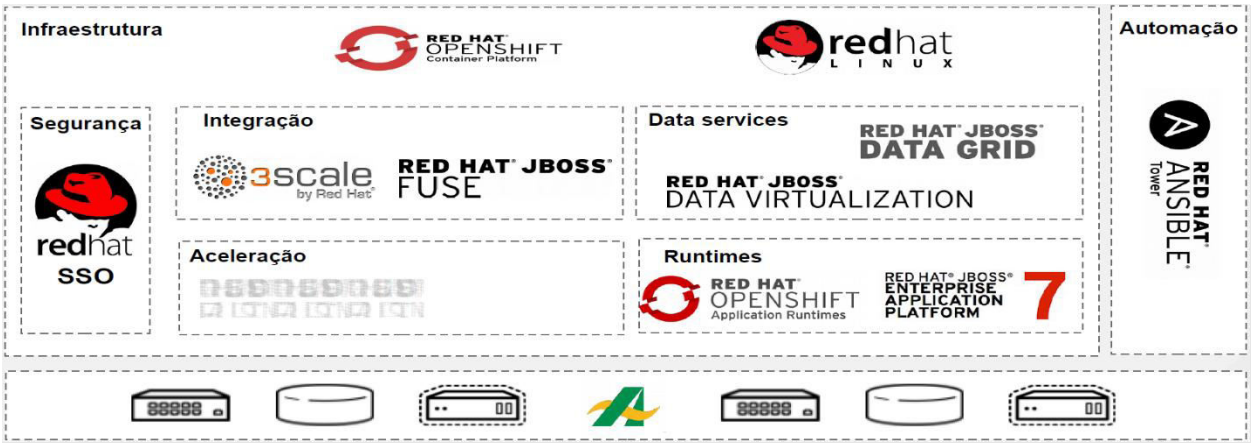
Dando seguimento ao PDTI, o Banco da Amazônia, em 2018, com foco na modernização acima descrita, deu início à análise de implementação do Barramento de Serviços Corporativos baseado em microcontainers - base estruturante para permitir a transformação digital da instituição, aumentando sua competitividade.

Diante do exposto, a implementação do barramento corporativo de serviços baseado em microcontainers no ambiente computacional do Banco da Amazônia visa realizar a modernização da infraestrutura e aplicações do Banco da Amazônia, criando uma arquitetura baseada em microsserviços e containers, tornando a manutenção e evolução mais ágil, simples e adequada para a entrega das novas necessidades digitais da instituição, conforme desenho arquitetural tecnológico a seguir.

### 1.2 Arquitetura do Red Hat OpenShift Container Platform[4]

Atualmente o Banco possui uma infraestrutura On Premise, com camada de virtualização, e visa buscar com o Red Hat OpenShift Container Platform um novo nível de desenvolvimento e publicação de aplicações, com métodos de CI/CD com a implementação dos seguintes produtos:

- Red Hat Ansible Tower;
- Red Hat JBoss Data Virtualization;
- Red Hat Process Automation Manager;
- Red Hat Decision Manager;
- Red Hat 3Scale API Management Platform;
- Red Hat JBoss Fuse Integration Services;
- Red Hat JBoss Enterprise Application Platform;
- Red Hat Single-Sign-On.

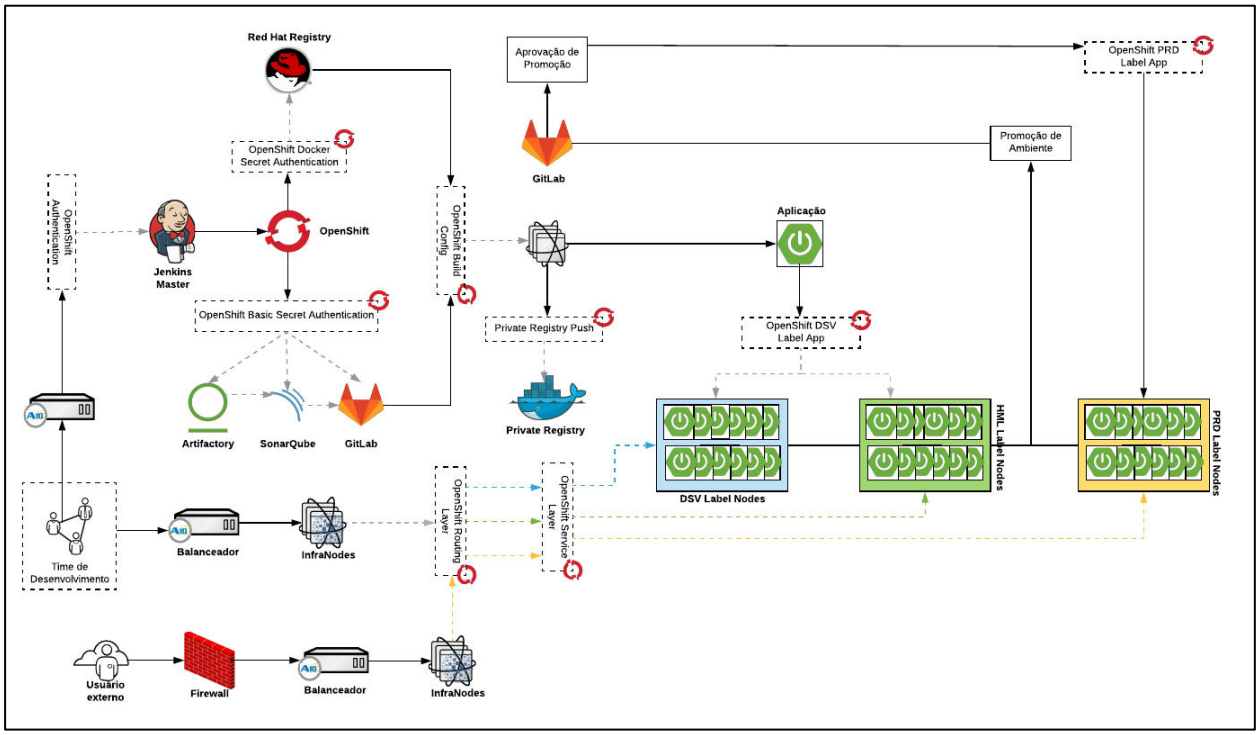



### 1.3 Arquitetura de Middleware

Após avaliação da arquitetura de sistemas atual, o Banco da Amazônia identificou diversos pontos que precisavam ser endereçados por uma arquitetura que oferecesse uma maior flexibilidade, robustez e confiabilidade as novas integrações e os novos sistemas que serão desenvolvidos na nova plataforma. Sendo assim, a Red Hat propôs uma arquitetura mais aderente às necessidades identificadas, para as quais se escolheu, também, os seguintes produtos:

[4] https://docs.openshift.com/container-platform/4.10/welcome/index.html

- 3scale;
- FUSE;
- Data Virtualization;
- BRMS;
- BPM Suite;
- RH-SSO;
- EAP.




## 1.4 Arquitetura do Open Finance[5,6,7]

O Open Finance é um modelo de compartilhamento de dados financeiros, autorizado pelos clientes, que envolve diferentes instituições financeiras e empresas, abrangendo não apenas bancos, mas também outros setores como seguros, investimentos, previdência e câmbio, expandindo a integração entre serviços financeiros, oferecendo maior flexibilidade e conectividade no uso de dados pessoais.

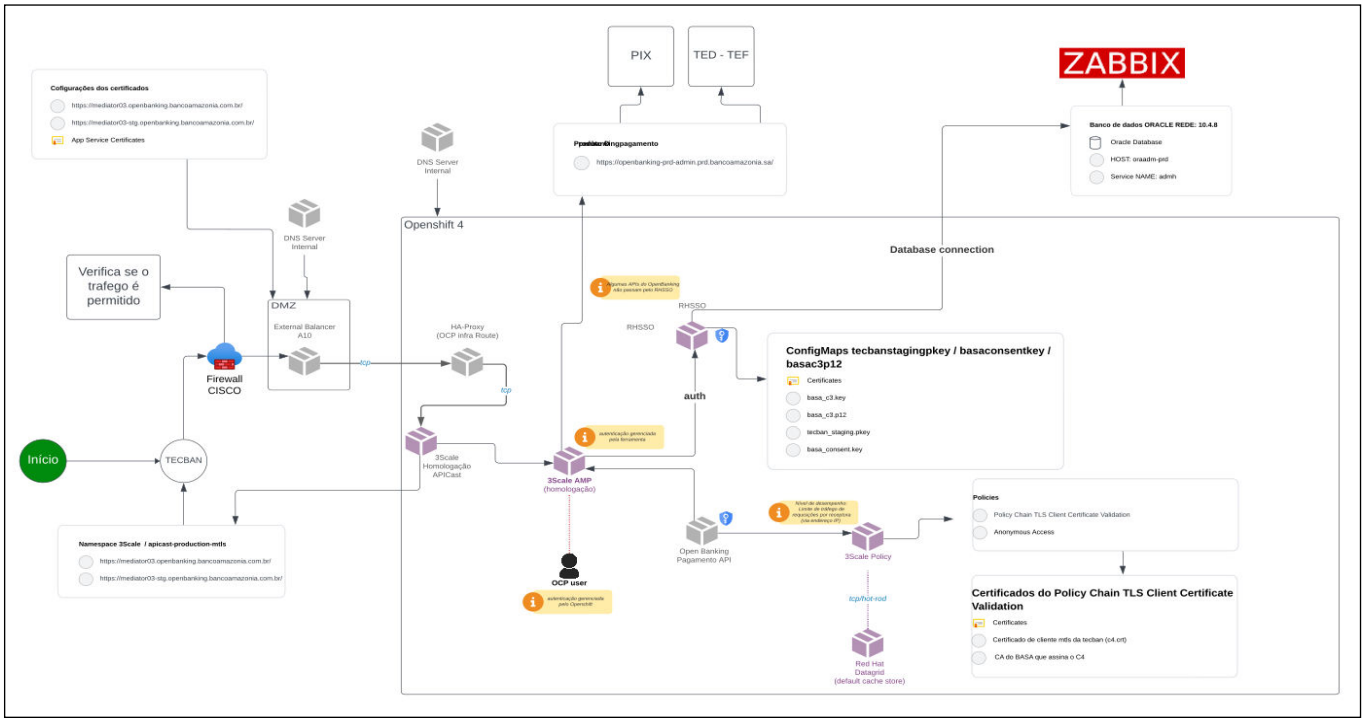
O principal objetivo do Open Finance é promover transparência, competitividade e inovação no setor financeiro, garantindo aos consumidores maior controle sobre seus dados e acesso a opções personalizadas de produtos e serviços. Por meio de APIs seguras e da integração de plataformas e sistemas de informação, o modelo busca assegurar operações seguras, ágeis e convenientes, permitindo que os clientes compartilhem suas informações de maneira protegida e conforme suas necessidades.

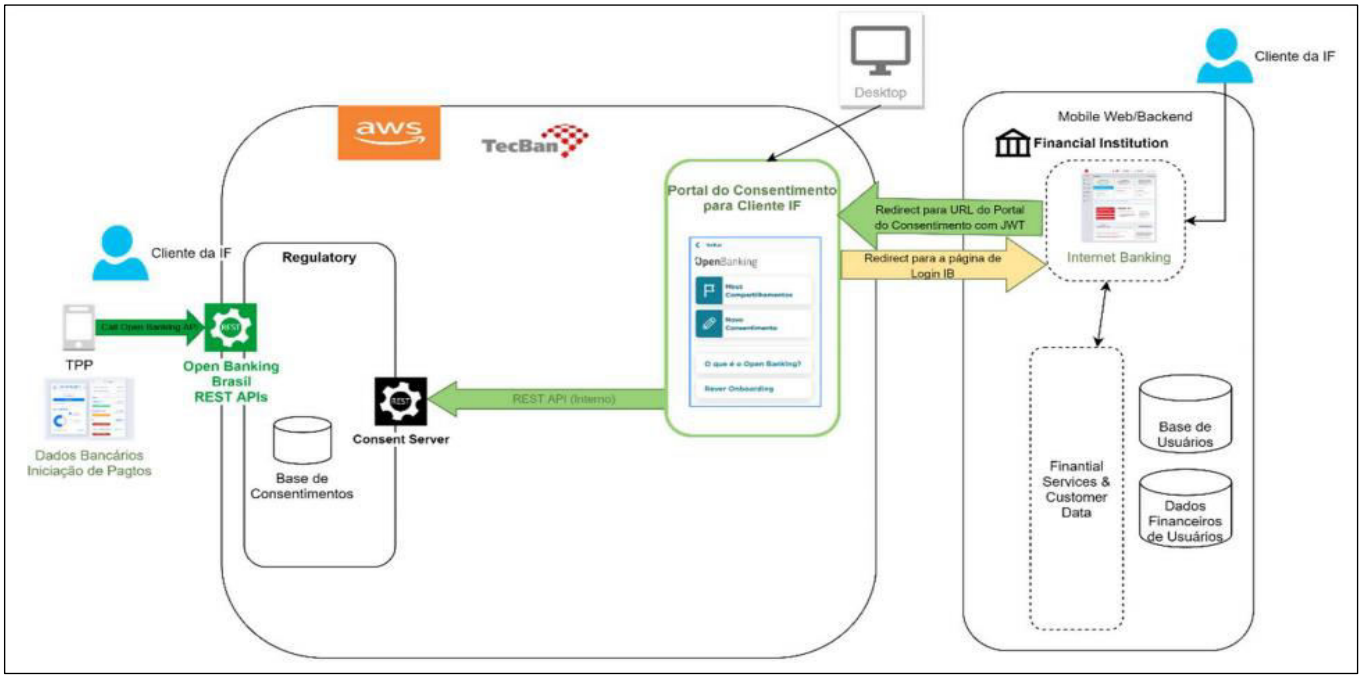
Diante desse cenário, o Banco da Amazônia, com sua equipe técnica e em parceria com as consultorias da Red Hat e da TECBAN, realizou estudos detalhados e propôs a implementação de uma solução que atenda às exigências do Open Finance Brasil, cuja definição arquitetural está representada a seguir, a qual descreve o modelo de implantação de ferramentas e produtos em ambiente produtivo, que é operado dentro do OpenShift, garantindo a infraestrutura necessária para o fornecimento seguro e eficiente dos serviços.

[5] https://www.bcb.gov.br/estabilidadefinanceira/openfinance

[6] https://github.com/OpenBanking-Brasil

[7] https://openfinancebrasil.org.br/


PIX
TED - TEF
ZABBIX
Configurações dos certificados
https://mediator03.openbanking.bancoamazonia.com.br/
https://mediator03-stg.openbanking.bancoamazonia.com.br/
App Service Certificates
DNS Server Internal
Openshift 4
Banco de dados ORACLE REDE: 10.4.8
Oracle Database
HOST: oraadm-prd
Service NAME: admin
Database connection
Verifica se o trafego é permitido
DMZ
External Balancer A10
Firewall CISCO
HA-Proxy (OCP infra Route)
RHSSO
auth
ConfigMaps tecbanstagingpkey / basaconsentkey / basac3p12
Certificates
basa_c3.key
basa_c3.p12
tecban_staging.pkey
basa_consent.key
Início
TECBAN
3Scale Homologação APICast
3Scale AMP (homologação)
OCP user
Open Banking Pagamento API
3Scale Policy
Policies
Policy Chain TLS Client Certificate Validation
Anonymous Access
Certificados do Policy Chain TLS Client Certificate Validation
Certificates
Certificado de cliente mtls da tecban (c4.crt)
CA do BASA que assina o C4
Red Hat Datagrid (default cache store)
Namespace 3Scale / apicast-production-mtls
https://mediator03.openbanking.bancoamazonia.com.br/
https://mediator03-stg.openbanking.bancoamazonia.com.br/


Desktop
Cliente da IF
aws
TecBan
Mobile Web/Backend
Financial Institution
Portal do Consentimento para Cliente IF
Redirect para URL do Portal do Consentimento com JWT
Redirect para a página de Login IB
Internet Banking
Cliente da IF
Regulatory
TPP
Open Banking Brasil REST APIs
Dados Bancários Iniciação de Pagtos
Consent Server
REST API (Interno)
Base de Consentimentos
Finantial Services & Customer Data
Base de Usuários
Dados Financeiros de Usuários

# PARTE V - PADRÕES ARQUITETURAIS PARA TECNOLOGIAS DE BIG DATA, ANALYTICS E DATA LAKE

## 1. COMPOSIÇÃO ARQUITETURAL DO BIG DATA, ANALYTICS E DATA LAKE

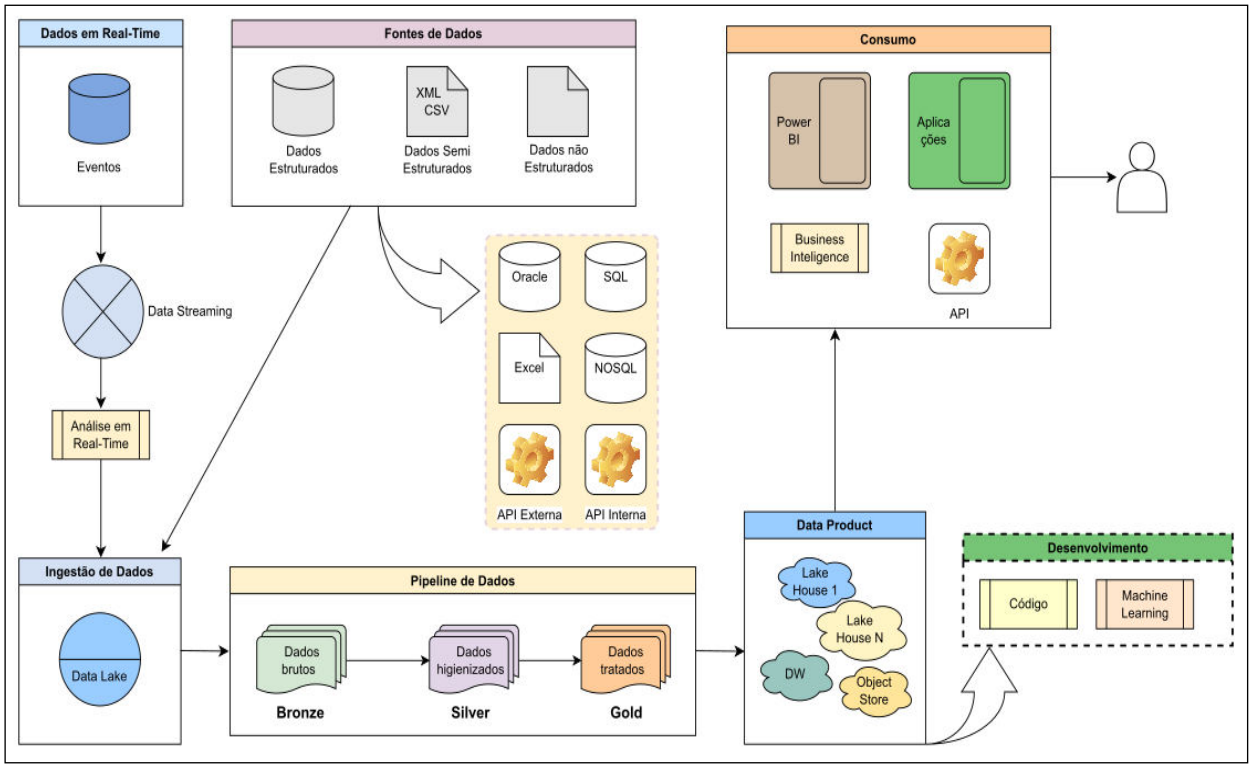
### 1.1 Visão e Desenho Geral

A modernização tecnológica do Banco da Amazônia, descrita em seu planejamento estratégico institucional, perpassa pela implementação de produtos digitais elencados no PDTI. Sendo assim, se torna necessária uma arquitetura referente a estruturação de serviços relacionados a Ciência de Dados (Big Data, Data Analytics, Machine Learning etc.) e Data Lake, que vai ao encontro da Visão do PEI 2023-2030, a qual cita a modernização tecnológica do banco e de seus canais de relacionamento.

Em resposta a isso, a inserção de tais recursos no ambiente tecnológico da instituição visa a possibilidade de alavancá-la a um patamar que eleve seu nível de performance e competitividade e, consequentemente, destaque e consolide seu grau de relevância e sua competência em meio a um mercado cada vez mais exigente e superaquecido.

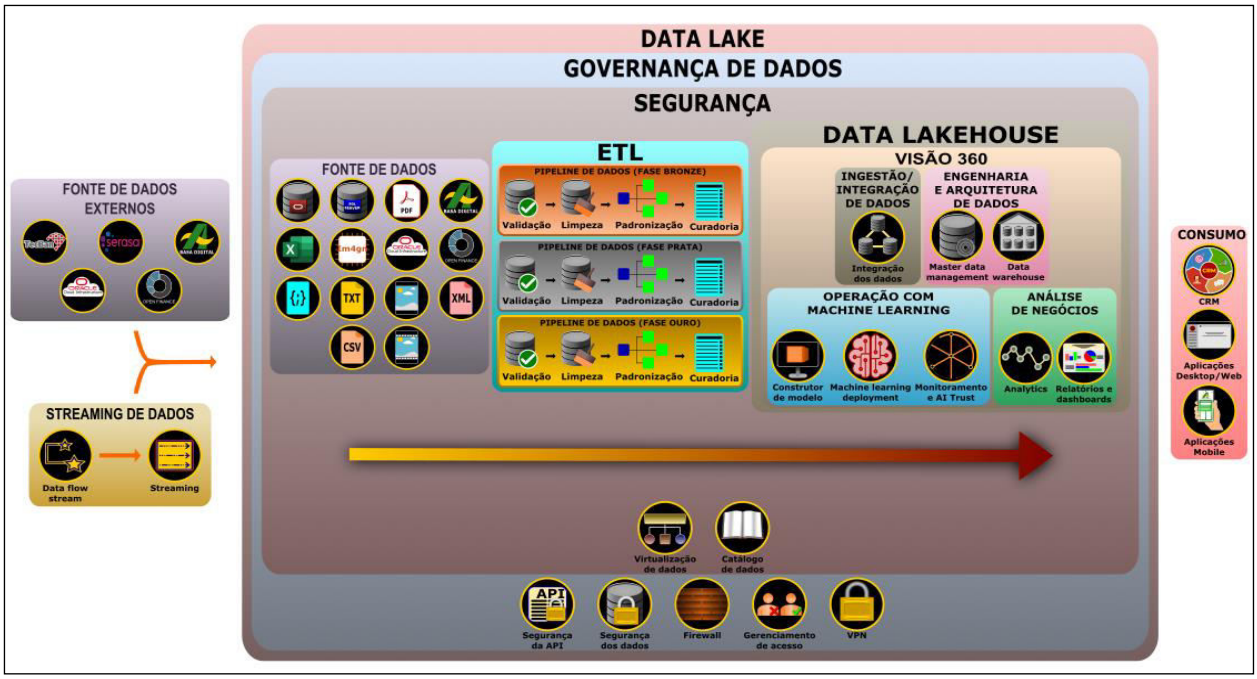
Como resultado obtém-se uma instituição que, diante dessa modernização, torna-se mais veloz e eficiente no quesito ação/interação em meio a cenários desafiadores e/ou, em algumas situações, desfavoráveis, mas tendo uma margem de risco que seja o mais mitigável quanto possível.

Logo abaixo, apresenta-se, de forma bem prática, uma visão geral da arquitetura como um todo e destacando suas principais camadas e funcionalidades.



## 1.2 Arquitetura do Big Data, Analytics e Data Lake

Visto que, atualmente, a instituição disponibiliza de uma estrutura de armazenamento on-premises e que, gradualmente, de acordo com a necessidade/oportunidade, migrará sua base de dados para armazenamento em nuvem, almeja-se que com a estrutura on-cloud seja possível a implementação de serviços tais como IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service) e Serverless (sem servidor) o que deixará todas as tarefas muito mais eficientes e práticas quando o assunto é manipulação de dados como um todo. No que diz respeito ao armazenamento voltado para Ciência de Dados, o Banco da Amazônia apresenta a seguinte estrutura visualizada de forma generalizada através do diagrama abaixo.

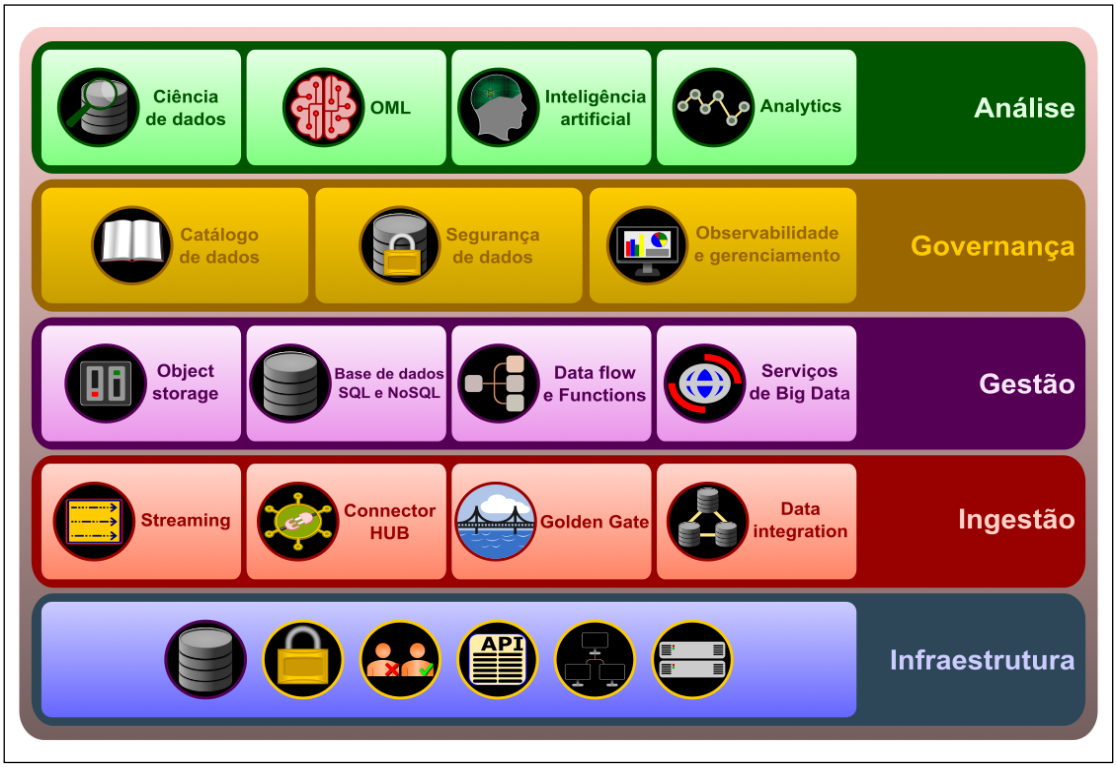


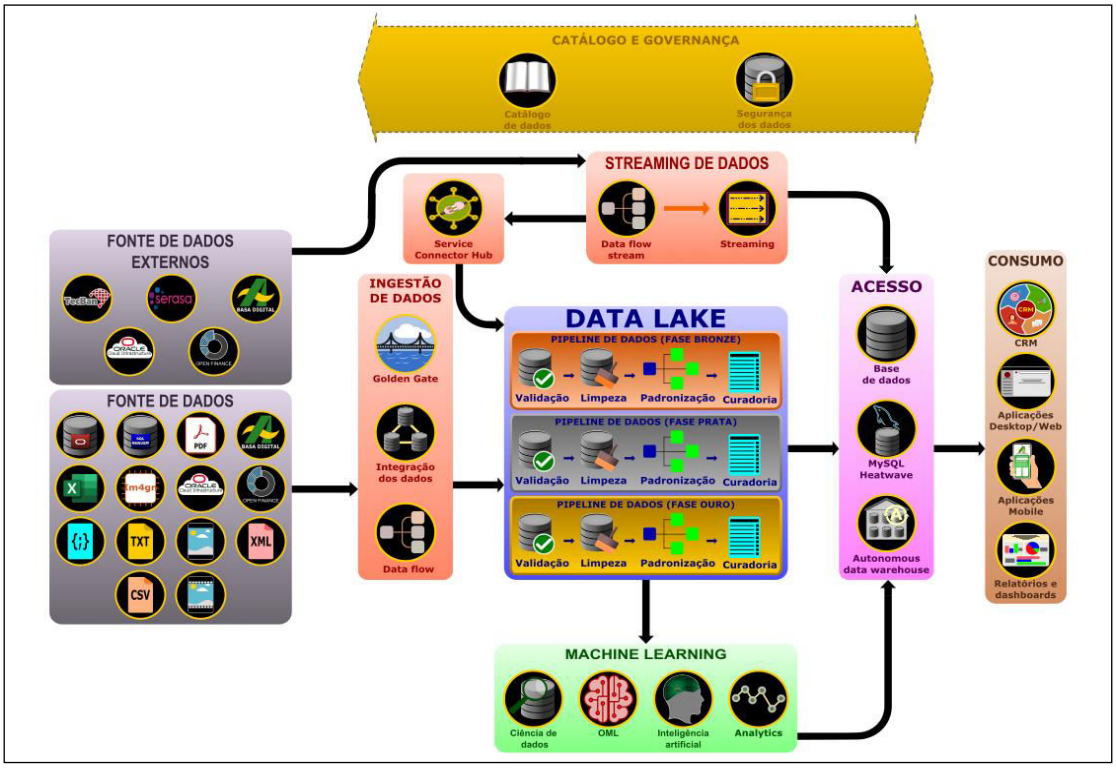

Descrição e características das camadas - da mais externa para a mais interna:

- **Primeira Camada - Data Lake Estruturado:** ambiente destinado ao armazenamento de dados dos mais variados formatos e em seu estado original.
- **Segunda camada - Governança de dados:** ferramentas para identificação, rotulação, caracterização e catalogação dos dados existentes no data lake;
- **Terceira Camada - Segurança de Dados:** ferramentas que proverão e garantirão segurança, confiabilidade e idoneidade dos dados coletados/acessados.
- **Quarta camada:** lida diretamente com os dados e possui as seguintes subcmadas:
  - **Fontes de Dados:** Origem das informações que serão coletadas. Podendo essas fontes serem de origem interna ou externa.
  - **ETL (extraction, treatment and loading):** ambiente de extração. tratamento e carregamento dos dados já estruturados e prontos para o uso em tomada de decisões.
  - **Visão 360 com IA:** Abordagem para analisar, entender e atender melhor às necessidades dos clientes. Estando dividida nas seguintes etapas:
    1. **Ingestão / Integração de Dados**: ferramentas que compreendem o processo de transformação e união dos dados previamente higienizados;
    2. **Engenharia e arquitetura de dados**: ferramentas para o processo de estruturação e design no que diz respeito ao armazenamento de dados a fim de garantir a disponibilidade para análise e uso posterior;

3. **MLOps (Machine Learning Operations):** ferramentas que combinam práticas de DevOps com aprendizado de máquina para gerenciar e implantar modelos de aprendizagem de maneira eficiente e eficaz;
4. **Analytics:** ferramentas para o processo de coleta e análise de dados para obter insights que ajudam na tomada de decisões estratégicas.

## 1.3 Arquitetura das Camadas



### 1.4.1 Analytics Cloud

A plataforma Oracle Analytics é um serviço nativo da nuvem que fornece os recursos necessários para abordar todo o processo analítico, incluindo ingestão e modelagem de dados, preparação e enriquecimento de dados, além de visualização e colaboração, sem comprometer a segurança e a governança. As tecnologias integradas de aprendizado de máquina e processamento de linguagem natural ajudam a aumentar a produtividade e a construir uma cultura orientada para análise nas organizações.

Referência: https://www.oracle.com/br/business-analytics/analytics-platform/

### 1.4.2 Data Catalog

O Oracle Cloud Infrastructure (OCI) Data Catalog é um serviço de gerenciamento de metadados que ajuda os profissionais de dados a descobrirem dados e oferecer suporte à governança de dados. Projetado especificamente para funcionar bem com o ecossistema Oracle, ele fornece um inventário de ativos, um glossário de negócios e um metastore comum para data lakes.

Referência: https://www.oracle.com/br/big-data/data-catalog/

### 1.4.3 Data Flow

O Oracle Cloud Infrastructure (OCI) Data Flow é um serviço totalmente gerenciado do Apache Spark que executa tarefas de processamento em conjuntos de dados extremamente grandes, sem infraestrutura para implementar ou gerenciar. Os desenvolvedores também podem usar o Spark Streaming para executar ETL na nuvem em seus dados de streaming produzidos continuamente. Isso permite a entrega rápida de aplicações porque os desenvolvedores podem se concentrar no desenvolvimento, não no gerenciamento da infraestrutura.

Referência: https://www.oracle.com/br/big-data/data-flow/

### 1.4.4 Data Integrator

O Oracle Data Integrator é uma plataforma abrangente que abrange todos os requisitos de integração de dados: de cargas de lote de alto volume e alto desempenho e processos de integração orientados a eventos e com alimentação tipo trickle feed até serviços de dados habilitados para SOA. O Oracle Data Integrator (ODI) 12c, a versão mais recente da oferta estratégica de Integração de Dados da Oracle, oferece produtividade superior ao desenvolvedor e melhor experiência para o usuário com uma interface declarativa, reprojetada e baseada em fluxo, além de integração mais profunda com o Oracle GoldenGate.

Referência: https://www.oracle.com/br/middleware/technologies/data-integrator.html

### 1.4.5 Data Safe

O Oracle Data Safe capacita as organizações a entenderem a sensibilidade dos dados, avaliar riscos de dados, mascarar dados confidenciais, implementar e monitorar controles de segurança, avaliar a segurança do usuário e monitorar a atividade do usuário, tudo em um único console unificado. Esses recursos ajudam a gerenciar os requisitos diários de segurança e conformidade de Oracle Databases, tanto on-premises quanto na nuvem.

Referência: https://www.oracle.com/br/security/database-security/data-safe/

### 1.4.6 Data Science

A Oracle Cloud Infrastructure (OCI) Data Science é uma plataforma totalmente gerenciada para que equipes de cientistas de dados criem, treinem, implantem e gerenciem modelos de machine learning usando o Python e ferramentas de código-fonte aberto.

Referência: https://www.oracle.com/br/artificial-intelligence/data-science/

### 1.4.7 DevOps

O serviço Oracle Cloud Infrastructure DevOps é uma plataforma completa de integração contínua/entrega contínua (CI/CD) para que os desenvolvedores simplifiquem e automatizem o ciclo de vida de desenvolvimento do software. O serviço OCI DevOps permite aos desenvolvedores e operadores desenvolverem, criar, testar e implantar software de forma colaborativa. Desenvolvedores e operadores têm visibilidade em todo o ciclo de vida de desenvolvimento, com um histórico de confirmação de origem através das fases de criação, teste e implantação.

Referência: https://www.oracle.com/br/devops/

### 1.4.8 Events

O Oracle Cloud Infrastructure (OCI) Events Service rastreia alterações feitas nos recursos usando eventos em conformidade com o padrão CloudEvents da Cloud Native Computing Foundation (CNCF). Os desenvolvedores podem responder a alterações feitas em tempo real, acionando o código com o Functions, gravando no Streaming ou enviando alertas usando o Notifications.

Referência: https://docs.oracle.com/pt-br/iaas/Content/Events/home.htm

### 1.4.9 Functions

O Oracle Cloud Infrastructure (OCI) Functions é um serviço de computação serverless que permite aos desenvolvedores criarem, executar e dimensionar aplicações sem gerenciar nenhuma infraestrutura. O Functions tem integrações nativas com outros serviços da Oracle Cloud Infrastructure e aplicações SaaS. Como o Functions se baseia no Projeto Fn de código-fonte aberto, os desenvolvedores podem criar aplicações que possam ser facilmente portadas para outros ambientes em nuvem e on-premises. O código baseado em Functions geralmente é executado por curtos períodos, sem estado e executado para um único propósito de lógica. Os clientes pagam apenas pelos recursos que usam.

Referência: https://docs.oracle.com/pt-br/iaas/Content/Functions/home.htm

### 1.4.10 Logging

Os clientes DevOps podem facilmente revisar dados de log, diagnosticar problemas e usar o mecanismo de regras para acionar o Functions serverless ou alertas com o Oracle Cloud Infrastructure (OCI) Logging. Desenvolvido com padrões abertos, o OCI Logging é uma plataforma intuitiva e centralizada para todos os tipos de logs, como auditoria, infraestrutura, banco de dados e aplicações, necessários para DevOps e conformidade de segurança.

Referência: https://www.oracle.com/br/devops/logging/

### 1.4.11 Logging Analytics

O Oracle Logging Analytics é uma solução em nuvem no Oracle Cloud Infrastructure que permite indexar, enriquecer, agregar, explorar, pesquisar, analisar, correlacionar, visualizar e monitorar todos os dados de log de seus aplicativos e da infraestrutura do sistema.

Referência: https://www.oracle.com/br/manageability/logging-analytics/

### 1.4.12 Monitoring

Serviço Oracle Cloud Infrastructure Monitoring para monitorar de forma ativa e passiva os recursos da nuvem usando as funcionalidades Métricas e Alarmes.

Referência: https://www.oracle.com/br/devops/monitoring/

### 1.4.13 Notification

O Oracle Cloud Infrastructure (OCI) Notifications é um serviço de publicação/assinatura (pub/sub) altamente disponível e de baixa latência que envia alertas e mensagens a Oracle Functions, Email, SMS e parceiros de entrega de mensagens, incluindo Slack, PagerDuty e ServiceNow. O serviço está integrado ao Oracle Identity and Access Management para acesso seguro e entrega cada mensagem, mesmo durante picos de tráfego.

Referência: https://www.oracle.com/br/devops/notifications/

### 1.4.14 Object Storage

O Oracle Cloud Infrastructure (OCI) Object Storage permite que os clientes armazenem com segurança qualquer tipo de dados em seu formato nativo. Com a redundância integrada, o Armazenamento de Objetos em OCI é ideal para criar aplicações modernas que requerem dimensionamento e flexibilidade, pois podem ser usados para consolidar várias fontes de dados para fins de análise, backup ou arquivamento.

Referência: https://www.oracle.com/br/cloud/storage/object-storage/

### 1.4.15 Apex

O Oracle APEX é a plataforma de aplicativos corporativos de baixo código mais popular do mundo, que permite criar aplicativos web e móveis seguros e escalonáveis, com recursos de classe mundial, que podem ser implantados em qualquer lugar - na nuvem ou no local.

Referência: https://apex.oracle.com/pt-br/

### 1.4.16 Advanced Analytics

Os desenvolvedores Oracle oferecem recursos avançados de análise, incluindo mineração de dados integrada para análise preditiva e integração com a linguagem estatística R. Especialistas em estatística, aprendizado de máquina e pesquisa operacional, fornecem ferramentas para cientistas e analistas de dados. A solução suporta todo o fluxo de trabalho analítico, com interface intuitiva, algoritmos escaláveis no banco de dados e funcionalidades para preparação e gestão de modelos. A integração com R permite executar programas em grandes conjuntos de dados armazenados, aproveitando a escalabilidade e o processamento eficiente do Oracle Database.

Referência: https://www.oracle.com/br/business-analytics/

### 1.4.17 Data Masking

Ferramenta que auxilia na descoberta do valor dos dados sem aumentar o risco, ao mesmo tempo que minimiza o custo de armazenamento. O Oracle Data Masking and Subsetting ajuda as organizações a obterem provisionamento de dados seguro e econômico para uma variedade de cenários, incluindo ambientes de teste, desenvolvimento e parceiros.

Referência: https://www.oracle.com/br/security/database-security/data-masking/

# ANEXO I - DOCUMENTAÇÃO DA ARQUITETURA DE SOFTWARE

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



**Arquitetura Corporativa**
Introdução
Padrões Arquiteturais
Políticas de Segurança
Configuração e Governança
Fluxo CI/CD
APIs REST

**Padrões de Desenvolvimento**
Java
Web
Nomenclatura para Banco de Dados
Configuração OpenShift

# Arquitetura de Software

On this page
Visão Geral
Padrões e Diretrizes
Linguagens de Programação
Servidores de Aplicação
Armazenamento de Dados
Interface do Usuário
Autenticação e Autorização
API Gateway
Controle de Versão de Códig...
Orquestração de Contêineres
CI/CD
Instalação e Deployment de ...

## Visão Geral

Para a maioria das aplicações, a arquitetura de software do Banco da Amazônia é robusta e escalável, suportando operações críticas. É composta por uma série de componentes interconectados, encapsulados em serviços RESTful distribuídos em um ambiente de computação em nuvem, utilizando OpenShift para orquestração de contêineres.

O serviço central de autenticação e autorização é baseado em OAuth 2.0 e JWT para segurança. Para controle de acesso interno. Além disso, o Banco da Amazônia utiliza o API Gate 3Scale para gerenciamento de APIs, garantindo segurança, controle de acesso e monitoramento.

Os dados são armazenados em bancos de dados Oracle, com opção para Red Hat Datagrid para grandes caches em memória. A empresa possui subscrição Red Hat, oferecendo suporte e acesso a ferramentas adicionais para desenvolvedores através do Basa.

Além das tecnologias e frameworks padronizados, também mantemos um conjunto de aplicações legadas desenvolvidas em outras linguagens de programação e tecnologias não padronizadas. Essas aplicações são integradas com os demais sistemas, garantindo a continuidade dos serviços e a interoperabilidade entre as diferentes soluções tecnológicas adotadas ao longo do tempo.

## Padrões e Diretrizes

A arquitetura segue um conjunto de padronizações para tornar o ambiente homogêneo, previsível e de fácil manutenção. Por isso, adotamos um conjunto limitado de tecnologias e frameworks, garantindo consistência e simplicidade. No entanto, essa lista é expandida conforme novas necessidades de negócio surgem, permitindo a adaptação e evolução contínua do ambiente.

Caso sejam adotadas, no desenvolvimento de novos sistemas internos, tecnologias ou frameworks que estejam fora de conformidade com os padrões estabelecidos pela Coordenadoria de Arquitetura (COARQ), esta não impedirá sua utilização, porém não fornecerá suporte ou assistência para a integração dessas soluções nos processos automatizados atualmente em vigor, como pipelines de CI/CD, monitoramento ou automações existentes.

A responsabilidade pela manutenção, evolução e operação de sistemas que utilizem tais tecnologias recairá integralmente sobre o setor requisitante. É fundamental que as tecnologias e frameworks adotados estejam em alinhamento com as diretrizes normativas para assegurar a interoperabilidade, eficiência e segurança do ecossistema de TI.

Em casos excepcionais, quando autorizados pelas instâncias superiores, se as soluções forem implementadas em ambientes on premise, a empresa contratada será integralmente responsável por todo o suporte necessário, incluindo a provisão de licenças de software, links de comunicação, componentes de infraestrutura, ambientes computacionais, entre outros.

Além disso, quaisquer necessidades ou configurações adicionais relacionadas a ambiente ou infraestrutura deverão ser obrigatoriamente alinhadas, acordadas e validadas previamente com a GPROD. A adoção de soluções fora dos padrões pode implicar em maior complexidade e custos operacionais, sendo responsabilidade do setor requisitante garantir sua plena funcionalidade e operação.

## Linguagens de Programação

O Banco da Amazônia adota as seguintes linguagens de programação para desenvolvimento de novos sistemas:

- Java
- Javascript/Typescript

## Servidores de Aplicação

Os servidores de aplicação são componentes essenciais da infraestrutura de Tecnologia da Informação (TI) do Basa, responsáveis por hospedar e fornecer aplicativos aos usuários finais. Eles desempenham um papel crucial na disponibilização, escalabilidade e desempenho das aplicações desenvolvidas pela organização.

Há comumente duas formas principais de hospedar aplicações desenvolvidas internamente:

- **JBoss EAP:**
  Utilizado para hospedar aplicações legadas desenvolvidas em Java EE.
- **Aplicações Standalone:**
  Aplicações desenvolvidas em *Spring Boot* para diferentes finalidades, como aplicações web, integrações, jobs, dentre outros.

Os servidores de aplicação JBoss EAP são tradicionalmente implantados em Máquinas Virtuais (VMs), enquanto as aplicações Spring Boot são disponibilizadas via OpenShift. Contudo, algumas aplicações para JBoss EAP foram migradas para OpenShift.

## Armazenamento de Dados

### Banco de Dados Relacional

O Banco da Amazônia faz uso exclusivo de bancos de dados *Oracle* para armazenamento e gerenciamento de dados para soluções de software desenvolvidas internamente, disponível na versão `19c`.

A coordenadoria de infraestrutura suporta o SQL Server apenas para soluções adquirida de terceiros já em uso. Devido aos custos de licença, o SQL Server não deve ser utilizado para novas aplicações. A versão suportada atualmente é a `2019`

### Armazenamento NoSQL

Recomenda-se o uso de *Red Hat Datagrid* para lidar com demandas específicas de cache em memória de grandes volumes de dados, proporcionando alta performance e escalabilidade para determinadas aplicações que requerem acesso *rápido* a conjuntos de dados em tempo real.

Para pequenos volumes de dados voláteis, recomenda-se o uso do Redis, um banco de dados em memória de código aberto.

▸ **Cuidados com a Otimização Prematura**

## Interface do Usuário

O componente de frontend é responsável pela apresentação da interface do usuário e pela interação direta com os usuários finais.

Para interfaces de usuário web, são utilizadas tecnologias modernas, como Vue.js e Quasar Framework, proporcionando uma experiência moderna e responsiva. Recomenda-se o uso de TypeScript para garantir a segurança e a manutenibilidade do código.

## Autenticação e Autorização

O *RH-SSO* (Red Hat Single Sign-On) é o serviço central de autenticação e autorização, responsável por garantir a segurança dos dados e controlar o acesso às diferentes partes do sistema, tanto para servidores de aplicação como interfaces web. Este serviço é disponibilizado via protocolo OAuth 2.0 e JWT (JSON Web Tokens), fornecendo uma camada de segurança robusta e flexível.

Em conjunto com o RH-SSO, o *AmazonSegu* é utilizado para controle de acesso interno mais granular, onde políticas de acesso dos sistemas são baseadas em funções e lotações departamentais dos usuários.

## API Gateway

O API Gateway *3Scale* gerencia e expõe APIs para clientes externos, fornecendo controle de acesso, monitoramento de tráfego e análise de métricas. O 3Scale oferece uma plataforma robusta para gerenciar políticas de autenticação, autorização, limitação de taxa e monetização de APIs.

## Controle de Versão de Código-Fonte

O GitLab é uma plataforma central para controle de versão, colaboração e gestão de projetos de software. Sua integração permite que equipes colaborem eficientemente, rastreiem mudanças, revisem solicitações de pull e gerenciem o ciclo completo de desenvolvimento de software.

## Orquestração de Contêineres

O OpenShift é uma plataforma de contêineres baseada em Kubernetes, projetada para oferecer uma infraestrutura de nuvem interna altamente escalável e automatizada.

As aplicações são encapsuladas em contêineres Docker e implantadas como Pods em nós de computação gerenciados pelo OpenShift. Utilizando recursos como Deployments, Services e Routes, o OpenShift simplifica o processo de implantação e oferece uma maneira eficiente de expor serviços para usuários finais.

O OpenShift oferece recursos de autoscaling baseados em métricas como uso de CPU e memória, permitindo que os Pods sejam escalados automaticamente para lidar com picos de tráfego. Além disso, o balanceamento de carga distribui o tráfego entre os Pods disponíveis, garantindo um uso eficiente dos recursos e uma experiência consistente para os usuários finais.

## CI/CD

O *Continuous Integration/Continuous Deployment* (CI/CD) é uma prática fundamental na engenharia de software, visando automatizar e agilizar o processo de desenvolvimento, teste e entrega de software. No centro dessa prática, adotada no Basa, estão ferramentas como *GitLab CI/CD*, *SonarQube*, *Docker* e *Artifactory*.

O CI/CD está implementado utilizando o GitLab *CI/CD* como orquestrador do pipeline para compilar, testar e implantar aplicações. Para mais detalhes de funcionamento e configuração do GitLab CI/CD, consulte a documentação oficial.

## Instalação e Deployment de Softwares

Primeiramente, é importante diferenciar aplicações produzidas internamente de softwares comprados ou soluções externas/integradoras. As aplicações internas são desenvolvidas por colaboradores do Banco ou terceiros, atendendo às demandas do setor de projetos. Elas são disponibilizadas preferencialmente através de contêiners, gerenciados por plataformas como Kubernetes ou OpenShift, o que simplifica a implantação e potencializa a escalabilidade, ou em servidores de aplicação JBoss EAP para aplicações legadas.

Softwares comprados ou soluções externas/integradoras requerem instalação em VMs dedicadas para assegurar a aderência às políticas de backup, recuperação de desastres (DR) e proporcionar flexibilidade para atualizações sem o risco de dependência de fornecedor. Portanto, essas soluções devem ser compatíveis com os componentes de infraestrutura apoiados pela equipe interna do Banco, como bancos de dados Oracle, servidores de aplicação JBoss e sistemas operacionais como RHEL (Red Hat Enterprise Linux).

Se houver decisão de instalar soluções adquiridas não compatíveis com os componentes de infraestrutura suportados ou fora de conformidade com as diretrizes descritas acima, as coordenadorias de arquitetura e infraestrutura não proibirão seu uso, mas também não endossarão ou darão suporte a essas soluções, sendo que a responsabilidade por sua manutenção e operação será do setor requisitante. É crucial que todas as implementações estejam em conformidade com os padrões e requisitos normatizados para assegurar a integridade e eficiência do ambiente de TI.

Next page
Introdução

Created by BASANO-ASE-PDF

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



Arquitetura Corporativa
Introdução
Padrões Arquiteturais
Políticas de Segurança
Configuração e Governança
Fluxo CI/CD
APIs REST

Padrões de Desenvolvimento
Java
Web
Nomenclatura para Banco de Dados
Configuração OpenShift

# Padrões Arquiteturais

Os padrões arquiteturais fornecem diretrizes e melhores práticas para projetar sistemas de software robustos, escaláveis e de fácil manutenção. Este capítulo identifica e explica os padrões arquiteturais utilizados na arquitetura do sistema, justificando suas escolhas e demonstrando como contribuem para a eficácia geral da solução.

## Nomenclatura de Projetos

Os nomes dados aos projetos tem um papel importante na organização e identificação dos sistemas, pois é através deles que ferramentas de versionamento, CI/CD, documentação e outros serviços identificam, organizam e configuram automaticamente os artefatos de software.

Para criação do projeto no GitLab, utilize nomes em formato *kebab-case*, considerando os seguintes padrões definidos por tipo de projeto:

> **Dica**
> kebab case combina palavras em caixa-baixa separados por hífen. Exemplo: **pagamento-tarifas**

- **APIs**: prefixados com `api-`
  Ex.: `api-credito-fomento`
- **Projetos Web**: prefixados com `web-`
  Ex.: `web-credito-fomento`
- **APPs Mobile**: prefixados com `app-`
  Ex.: `app-credito-fomento`
- **Jobs**: prefixados com `job-`
  Ex.: `job-credito-fomento`
- **Migration de Banco de Dados**: Utilizar o nome do schema do Banco de Dados
  Ex.: `dbxflow_ bg_prd_gc`
- **Demais Projetos**: sem prefixos ou sufixos

## Versionamento de Sistemas

O versionamento dos projetos deve seguir o padrão "*major.minor.patch*", inspirado no Semantic Versioning

- `Major Version`: O incremento deve ser manual, destacando as mudanças significativas no software. Essas alterações geralmente são substanciais, modificando muitos módulos, interfaces do usuário e possivelmente rompendo a compatibilidade retrativa;
- `Minor Version`: O incremento deve ser feito manualmente, sinalizando a adição de novas funcionalidades, enquanto preserva a compatibilidade com versões anteriores;
- `Patch`: O incremento pode ser automatizado e deve sinalizar a liberação de uma nova build, geralmente destinada à correção de bugs e operações similares.

Podemos adicionar o sufixo *SNAPSHOT* para sinalizar versões de desenvolvimento de sistemas. O incremento do snapshot é realizado adicionando um quarto conjunto numérico. Por exemplo: *1.2.3.10-SNAPSHOT*.

## Amazônia Framework

O *Amazônia Framework* é uma coleção interna de bibliotecas e ferramentas baseadas em tecnologias padrões da arquitetura do Banco da Amazônia. Tem por objetivos:

1. Agilizar o desenvolvimento de sistemas no Banco da Amazônia
2. Disponibilizar soluções testadas e homologadas para requisitos não-funcionais recorrentes, já integradas com soluções de infraestrutura presentes.
3. Homogeneizar a arquitetura dos sistemas bancários e administrativos, facilitando a manutenção e o processo de repasse de conhecimento entre os fornecedores e analistas

Conforme a arquitetura do Banco evolui e novas plataformas de desenvolvimento são incluídas, podem ser implementados novos módulos do Amazônia Framework para atender a cada plataforma.

## Padrão MVC

O padrão MVC (Model-View-Controller) é comumente utilizado em aplicações web para separar a aplicação em três camadas interconectadas. Isso facilita a manutenção do código, permitindo que cada componente seja desenvolvido e testado independentemente. Além disso, o MVC promove a reutilização de código e a escalabilidade da aplicação.

## Arquitetura em Camadas

A arquitetura em camadas divide o sistema em camadas distintas, cada uma responsável por uma função específica. Por exemplo, uma aplicação web pode ter camadas de apresentação, lógica de negócios e acesso a dados. Esta abordagem promove a separação de preocupações, facilitando a manutenção, a escalabilidade e a reutilização de código.

O Domain-Driven Design (DDD) é utilizado como modelo para organização das camadas arquiteturais. No contexto de DDD, as camadas refletem os conceitos do domínio de negócios, promovendo uma arquitetura mais alinhada com as necessidades e linguagem da empresa. As camadas geralmente incluem:

- **Camada de Apresentação:** Responsável por fornecer interfaces de usuário para interagir com o sistema. Esta camada pode incluir interfaces web, APIs RESTful ou interfaces de linha de comando.
- **Camada de Aplicação:** Aqui reside a lógica de aplicação que coordena as interações entre as diferentes partes do sistema. Esta camada geralmente implementa os casos de uso do sistema e é responsável por orquestrar os serviços do domínio, integrações externas, validações, persistência, dentre outros.
- **Camada de Domínio:** Esta é a camada central do sistema, onde residem os modelos de domínio, regras de negócio e lógica de domínio. Aqui, os conceitos do negócio são modelados em classes e entidades de dados que refletem a linguagem ubíqua da empresa.

  > **Informações Extras**
  > A "linguagem ubíqua" é um vocabulário compartilhado em uma equipe de desenvolvimento e stakeholders, consistindo de termos e conceitos comuns utilizados de forma consistente em todo o projeto, independentemente do papel de cada pessoa.
- **Camada de Infraestrutura:** Responsável por fornecer suporte técnico para as camadas superiores. Isso inclui acesso a banco de dados, comunicação de rede, logging, etc. A camada de infraestrutura geralmente encapsula detalhes técnicos e é projetada para ser facilmente substituível e escalável.

Embora o DDD e o MVC sejam conceitos distintos, eles podem ser combinados e utilizados em conjunto para desenvolver sistemas de software mais robustos e com uma melhor representação do domínio do problema.

No contexto do desenvolvimento de aplicações web, por exemplo, é comum usar o MVC para estruturar a aplicação e separar as responsabilidades de apresentação, enquanto se aplica os princípios do DDD para modelar o domínio do problema e implementar a lógica de negócios de forma eficaz dentro do Modelo.

> **Dica**
> Nem sempre é necessário aplicar o DDD caso a aplicação seja simples e não possua um domínio complexo. No entanto, o MVC é a abordagem comum para estruturar aplicações web, independentemente da complexidade do domínio.

Em uma arquitetura em camadas, cada camada comunica-se com as camadas adjacentes de acordo com um conjunto de regras bem definido:

1. **Acesso direto:** Cada camada tem permissão para acessar diretamente apenas a camada imediatamente inferior e superior.
2. **Interfaces bem definidas:** Cada camada expõe interfaces bem definidas para as camadas adjacentes. Essas interfaces especificam os métodos e dados que estão disponíveis para acesso. Dessa forma, as camadas podem se comunicar apenas através dessas interfaces, mantendo uma separação clara de responsabilidades.
3. **Inversão de Controle (IoC):** A inversão de controle é usada para permitir que as camadas superiores forneçam serviços para as camadas inferiores. Em vez de a camada inferior chamar diretamente a camada superior, a camada superior pode "injetar" dependências na camada inferior, fornecendo funcionalidades necessárias. Isso permite uma maior flexibilidade e desacoplamento entre as camadas.
4. **Eventos e mensagens:** As camadas podem se comunicar por meio de eventos ou mensagens. Por exemplo, quando algo importante acontece em uma camada, ela pode emitir um evento que outras camadas podem ouvir (ex.: via listener) ou mensagem (ex.: chamada de função) conforme necessário.

## Arquitetura Event-Driven

A arquitetura event-driven é baseada na troca de eventos assíncronos entre componentes do sistema. Isso permite que os componentes reajam a eventos em tempo real, facilitando a integração entre sistemas distribuídos e promovendo a escalabilidade, flexibilidade e baixo acoplamento entre si. Padrões event-driven incluem mensageria assíncrona, como MQ, e padrões de publicação/assinatura, como MQTT.

Por exemplo, em uma aplicação de comércio eletrônico, a adição de um produto ao carrinho pode gerar um evento "produto adicionado ao carrinho". Este evento é então propagado para outros componentes interessados, como o componente de cálculo de preço, que pode recalculá-lo, ou o componente de análise de dados, que pode registrar a atividade do usuário para fins de análise de mercado.

## Arquitetura REST

A arquitetura REST (Representational State Transfer) é um estilo de arquitetura de software para sistemas distribuídos, que define um conjunto de restrições para projetar serviços web, proposta por Roy Fielding em sua tese de doutorado em 2000. Ela promove a interoperabilidade entre sistemas distribuídos, permitindo que os clientes acessem e manipulem recursos de forma uniforme e eficiente.

Alguns dos princípios-chave do REST incluem:

1. **Recursos**: Os recursos são entidades abstratas que podem ser acessadas através de identificadores únicos, como URLs. Cada recurso deve ter uma representação, que pode ser um documento HTML, XML, JSON, ou qualquer outro formato.
2. **Interfaces uniformes**: O REST promove a padronização das interfaces, utilizando métodos HTTP padrão, como GET, POST, PUT e DELETE, para realizar operações em recursos.
3. **Estado do cliente e do servidor**: A comunicação no REST é sem estado, o que significa que cada solicitação do cliente para o servidor deve conter toda a informação necessária para entender e processar a requisição. Isso facilita a escalabilidade do sistema.
4. **Sistema em camadas**: O REST permite a utilização de sistemas em camadas, onde cada componente (como servidores intermediários, caches, balanceadores de carga, etc.) pode ser implantado de forma independente, melhorando a escalabilidade e a segurança do sistema.
5. **Cache**: O REST aproveita os mecanismos de cache HTTP para melhorar o desempenho e a eficiência da comunicação entre cliente e servidor.

## Padrão SOA

A arquitetura orientada a serviços (SOA) organiza a funcionalidade em serviços independentes, acessíveis remotamente por meio de interfaces definidas, promovendo a reutilização de serviços e facilitando a integração entre sistemas heterogêneos.

A combinação de APIs REST e SOA possibilita uma arquitetura flexível e adaptável, onde as APIs REST padronizadas expõem os serviços para os clientes, permitindo acesso eficiente e transparente às funcionalidades do sistema.

## MVVM (Model-View-ViewModel)

O padrão MVVM (Model-View-ViewModel) é uma arquitetura de design que separa a lógica de apresentação da interface do usuário (View) e da lógica de negócios e dados (Model), utilizando um intermediário chamado ViewModel para facilitar a comunicação entre eles. Seus componentes incluem:

1. **Model**: Representa os dados e a lógica de negócios da aplicação.
2. **View**: Responsável pela apresentação da interface do usuário.
3. **ViewModel**: Atua como um intermediário entre a View e o Model, traduzindo dados do Model para a View e vice-versa.

Em uma aplicação de lista de tarefas, o Model poderia ser uma lista de objetos de tarefa com métodos para adicionar, remover e atualizar tarefas. A View seria a interface do usuário que exibe a lista de tarefas, e o ViewModel seria responsável por fornecer os dados das tarefas para a View e manipular as interações do usuário, como adicionar ou excluir tarefas.

No contexto de integração de aplicações web com APIs REST, o ViewModel desempenha um papel crucial na comunicação entre a View e a API. Ele seria responsável por fazer solicitações à API REST para recuperar ou modificar os dados do Model, e em seguida, atualizar a View com os dados obtidos.

O Vue.js é um framework JavaScript que facilita a implementação do padrão MVVM em aplicações web. Ele fornece recursos para criar componentes reativos que podem ser facilmente vinculados a dados do ViewModel e atualizados automaticamente conforme esses dados mudam. O Vue.js também oferece suporte para manipulação de eventos e gerenciamento de estado, o que simplifica o desenvolvimento de aplicações baseadas em MVVM.

## Testes Unitários

Os testes unitários são uma prática fundamental na engenharia de software para garantir a qualidade e a confiabilidade do código. Eles verificam o comportamento de unidades individuais de código, como funções, métodos ou classes, isoladamente do restante do sistema. Isso permite identificar e corrigir problemas de forma rápida e eficiente, promovendo a manutenção do código e a evolução do sistema.

Garantem que cada unidade de código funcione conforme o esperado, produzindo resultados corretos para uma variedade de entradas. Esses testes são geralmente escritos pelos próprios desenvolvedores e executados de forma automatizada durante o ciclo de desenvolvimento de software.

Abaixo seguem algumas práticas recomendadas para escrever testes unitários eficazes:

1. **Teste apenas uma unidade de código por vez**: Um teste unitário deve se concentrar em testar uma única funcionalidade isoladamente. Isso ajuda a identificar problemas com precisão.
2. **Seja específico e conciso**: Cada teste deve verificar uma única condição específica. Evite testes genéricos que tentam cobrir vários cenários.
3. **Use nomes descritivos para os testes**: Os nomes dos testes devem ser claros e descritivos para que outros desenvolvedores entendam facilmente o que está sendo testado.
4. **Teste todos os casos de borda**: Certifique-se de testar não apenas os casos típicos, mas também os limites e condições extremas. Isso inclui testar entradas inválidas e valores extremos.
5. **Mantenha os testes independentes e isolados**: Os testes não devem depender uns dos outros. Cada teste deve ser independente e executável em qualquer ordem.
6. **Use ferramentas de assertiva apropriadas**: Escolha assertivas adequadas para verificar o comportamento esperado do código. Isso pode incluir assertivas específicas para igualdade, exceções, tipos, etc.
7. **Automatize os testes**: A automação dos testes é essencial para garantir que os testes sejam executados de forma consistente e repetível, sempre que houver alterações no código.
8. **Execute os testes regularmente**: Os testes devem ser executados com frequência, de preferência em cada compilação ou implantação, para identificar rapidamente problemas.
9. **Mantenha os testes atualizados**: À medida que o código é alterado, os testes devem ser atualizados para refletir essas mudanças. Caso contrário, os testes podem se tornar obsoletos e ineficazes.
10. **Analise a cobertura dos testes**: Use ferramentas de análise de cobertura para garantir que os testes cubram adequadamente o código-fonte. Isso ajuda a identificar áreas não testadas e aprimorar a qualidade dos testes.

On this page
Nomenclatura de Projetos
Versionamento de Sistemas
Amazônia Framework
Padrão MVC
Arquitetura em Camadas
Arquitetura Event-Driven
Arquitetura REST
Padrão SOA
MVVM (Model-View-ViewMo...
Testes Unitários



Created by BANCO AMAZÔNIA

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



**Arquitetura Corporativa**
Introdução
Padrões Arquiteturais
Políticas de Segurança
Configuração e Governança
Fluxo CI/CD
APIs REST

**Padrões de Desenvolvimento**
Java
Web
Nomenclatura para Banco de Dados
Configuração OpenShift

# Políticas de Segurança

On this page
Introdução
Separação de Ambientes
Código-Fonte
Autenticação e Autorização
Controle de Acesso (Segu)
Certificados Digitais
Análise de Segurança Estátic...
Tecnologias Utilizadas
Dependências Externas
Normativo de Segurança

## Introdução

A segurança é uma consideração fundamental em qualquer sistema de software, especialmente quando se trata de arquiteturas distribuídas e expostas a ambientes externos. Este capítulo aborda as políticas de segurança implementadas em nosso sistema, incluindo autenticação, autorização, controle de acesso e proteção contra vulnerabilidades.

## Separação de Ambientes

Os ambientes de rede e sistemas são separados em três categorias distintas - desenvolvimento, homologação e produção - a fim de garantir maior controle da infraestrutura de tecnologia da informação. Esta abordagem oferece uma série de benefícios significativos em termos de segurança, destacando-se os seguintes pontos:

1. **Isolamento de Ambientes**: A separação entre os ambientes de desenvolvimento, homologação e produção assegura seu isolamento mútuo. Isso limita a propagação de problemas, como bugs ou ataques de segurança, entre eles. Assim, um ataque bem-sucedido no ambiente de desenvolvimento não impacta diretamente a produção, reduzindo o potencial dano.

2. **Testes e Validação Seguros**: O ambiente de homologação é um local seguro para testar e validar novos recursos, atualizações e patches de segurança antes da implantação em produção, minimizando o risco de vulnerabilidades de segurança ou problemas de compatibilidade que possam afetar a integridade da produção.

3. **Limitação de Acesso e Controle de Permissões**: Cada ambiente pode possuir políticas de acesso e permissões distintas, adequadas às funções dos usuários. Por exemplo, somente pessoal autorizado tem acesso ao ambiente de produção, enquanto os desenvolvedores possuem permissões mais extensas no ambiente de desenvolvimento para facilitar a criação e depuração de aplicativos.

4. **Monitoramento e Auditoria Aprimorados**: A separação dos ambientes facilita o monitoramento e a auditoria granular das atividades, permitindo uma detecção ágil de ações suspeitas ou anômalas. Isso simplifica a investigação e resposta a incidentes de segurança, ao tornar mais fácil a identificação de possíveis fontes de comprometimento.

5. **Resiliência e Continuidade dos Negócios**: Ao manter ambientes separados, uma organização pode mitigar os impactos de falhas ou ataques em um ambiente específico. Isso pode ajudar a garantir a continuidade dos negócios, permitindo que outras partes da infraestrutura permaneçam operacionais, mesmo que um ambiente seja comprometido.

6. **Padrões de Desenvolvimento e Configuração Consistentes**: A separação dos ambientes promove padrões consistentes de desenvolvimento e configuração em toda a organização, minimizando erros humanos e configurações impróprias que possam gerar vulnerabilidades.

> **Dica**
> Na prática, utilize configuração externa para parametrizar o comportamento dos sistemas por ambiente. Isso permite que as configurações sejam alteradas sem a necessidade de recompilar o código.

## Código-Fonte

Os códigos-fonte deverão ser armazenados em ambiente seguro e controlado. Para isso, será utilizado o gerenciador de repositório GitLab, acessível a partir deste link.

### Gestão de Acesso no GitLab

Os códigos-fonte de sistemas ou de bibliotecas deverão ter seu acesso controlado e restrito aos colaboradores autorizados, a fim de se garantir sua integridade e confidencialidade. Caberá aos responsáveis do projeto gerir estes acessos.

Dada a responsabilidade de cada integrante e a classificação de acesso do GitLab, é definido que nos *projetos ou grupos de projetos*:

- **Terceirizados** devem apenas ter permissão de `developer`;
- **Analistas do Basa** devem ter permissão `developer` ou `maintainer`, dependendo da responsabilidade e experiência;
- **Gestores técnicos** devem ter permissão de `owner`
- **Pessoas não relacionadas diretamente** às atividades do projeto podem ter permissão de `reporter` apenas.

Sobre configuração de visibilidade dos projetos no GitLab:

- **Grupos de repositórios** devem ter visibilidade `internal`
- **Projetos** devem ter visibilidade `private`

### Bibliotecas Internas

Além de seus códigos-fonte serem armazenados adequadamente em repositórios seguros, as bibliotecas desenvolvidas internamente deverão ser empacotadas e distribuídas por gerenciadores de bibliotecas, assim evitando o uso de bibliotecas mantidas apenas na estação de trabalho do desenvolvedor. Os gerenciadores de bibliotecas disponível são:

- Artifactory, para bibliotecas Java
- Nexus, para bibliotecas Javascript

## Autenticação e Autorização

Para garantir a autenticação e autorização dos usuários, adotamos a solução RH-SSO (Red Hat Single Sign-On). O RH-SSO oferece uma maneira segura de gerenciar a identidade dos usuários, permitindo o acesso controlado aos recursos do sistema com base em suas permissões. Utilizamos mecanismos padrão de autenticação, como OAuth 2.0, para garantir a segurança das credenciais dos usuários durante o processo de login.

Há instâncias por ambiente do RH-SSO responsáveis pela gestão de acesso de sistemas internos, nas quais os usuários, incluindo funcionários, terceiros e contas de serviço, são sincronizados a partir do Active Directory (AD) da instituição. Estão acessíveis a partir os links abaixo:

- Ambiente de Desenvolvimento/Homologação
- Ambiente de Produção

> **Cuidado**
> O uso do HTTP Basic Authentication não é permitido para desenvolvimento de APIs.

▸ **Mais sobre OAuth 2.0**

▸ **Mais sobre Active Directory**

## Controle de Acesso (Segu)

O AmazonSegu é uma ferramenta para o controle de acesso e definição de políticas de segurança. Com ele, podemos definir regras granulares para acesso aos recursos de sistemas através de grupos de acesso criados pela composição das funções/cargos de funcionários e sua lotação departamental.

O AmazonSegu é acessível através dos links abaixo:

- Ambiente de Desenvolvimento/Homologação
- Ambiente de Produção

## Certificados Digitais

Certificados digitais são essenciais para a segurança na comunicação entre aplicativos e sistemas externos. Eles fornecem uma maneira de verificar a identidade de uma entidade e estabelecer uma conexão segura. Ao se comunicar com sistemas externos, um aplicativo pode precisar confiar em um certificado fornecido por esse sistema. Isso é comum em situações de comunicação SSL/TLS, onde o sistema externo apresenta um certificado para autenticar sua identidade.

O Banco da Amazônia utiliza certificados digitais autoassinados para garantir a segurança da comunicação entre sistemas internos. Esses certificados são gerados e gerenciados internamente e devem ser importados para os repositórios de certificados confiáveis dos aplicativos que precisam se comunicar com esses sistemas.

> **Dica**
> Para o ambiente do Basa, importe os certificados disponibilizados aqui para o seu repositório de certificados confiáveis, como por exemplo, o cacerts do Java ou o certs do Node.js.

## Análise de Segurança Estática (SAST)

Como parte de nossas práticas de desenvolvimento seguro, integramos análise de segurança estática em nosso pipeline de CI/CD (Continuous Integration/Continuous Deployment). Utilizamos o SonarQube para realizar análises estáticas de código fonte em busca de vulnerabilidades conhecidas e boas práticas de segurança. Isso nos permite identificar e corrigir potenciais falhas de segurança durante o desenvolvimento, reduzindo o risco de exposição a ataques. O SonarQube é acessível a partir deste link.

▸ **mais sobre SAST**

## Tecnologias Utilizadas

Nossa instituição adota uma abordagem estratégica na seleção de tecnologias, visando simplificar a Gestão de Configuração e fortalecer a segurança dos sistemas. Optamos por um conjunto enxuto, porém robusto, de ferramentas, incluindo Java com Spring Boot para o backend e Vue.js para o frontend. Essa escolha visa reduzir a complexidade operacional e promover a previsibilidade no processo de integração continua e entrega continua (CI/CD).

Outras tecnologias podem ser avaliadas para uso conforme surgem novas necessidades, desde que atendam aos critérios de segurança e conformidade, como também há possibilidade de desenvolvimento de novos módulos do Amazonia Framework para suportar demais linguagens ou plataformas.

## Dependências Externas

O código-fonte gerado internamente é apenas uma parte da base de código dos sistemas. Frameworks e bibliotecas de terceiros também devem passar certos critérios de avaliação, pois se forem vulneráveis ou obsoletos, provavelmente afetarão seu sistema.

Não há uma forma simples de garantir que as dependências de terceiros sejam seguras e confiáveis. No entanto, podemos adotar práticas recomendadas para minimizar o risco, como:

- Utilizar ferramentas de análise de vulnerabilidades para identificar e corrigir vulnerabilidades conhecidas em bibliotecas de terceiros.
- Verificar se as bibliotecas de terceiros são mantidas ativamente e têm uma comunidade de suporte ativa.
- Sempre que possível, atualizar para as versões mais recentes das bibliotecas de terceiros, que geralmente corrigem vulnerabilidades e bugs.
- Evitar uso de bibliotecas de terceiros modificadas ou não oficiais, pois elas podem conter código malicioso ou vulnerabilidades.

Novas bibliotecas externas podem ser utilizadas para desenvolvimento de sistemas no Basa sem aprovação prévia do time de arquitetura, a menos que substituam uma biblioteca em uso já consolidada.

## Normativo de Segurança

A GESEC (Gerência Executiva de Segurança Corporativa) é responsável por estabelecer normas, padrões e metodologias para a aquisição, desenvolvimento e manutenção de sistemas seguros.

Ela disponibiliza a *NP SEGURANÇA NO DESENVOLVIMENTO DE SISTEMAS*, que é o principal documento normativo que orienta as práticas de segurança adotadas em nossos sistemas. Este documento estabelece diretrizes para o desenvolvimento seguro de sistemas, incluindo requisitos de autenticação, autorização, controle de acesso, criptografia, auditoria e registro de eventos, entre outros aspectos de segurança.

Acesse a versão recente desta NP no repositório de normativos ou baixe a versão disponível neste link.



Generated by BRYAN_AGENT

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



**Arquitetura Corporativa**
Introdução
Padrões Arquiteturais
Políticas de Segurança
Configuração e Governança
Fluxo CI/CD
APIs REST

**Padrões de Desenvolvimento**
Java
Web
Nomenclatura para Banco de Dados
Configuração OpenShift

# Gestão de Configuração e Governança

Este documento descreve as configurações necessárias para o desenvolvimento de software no Banco da Amazônia. As configurações são necessárias para garantir a segurança e a rastreabilidade das atividades de desenvolvimento.

On this page
Solicitação de Infraestrutura ...
Documentação Geral e Wiki
Configurações do GitLab
Instalação de Certificados

## Solicitação de Infraestrutura para Projetos

Os documentos Plano de Arquitetura e Plano de Infraestrutura são obrigatórios para solicitar criação dos ambientes de deployment no OpenShift e devem ser criados na Wiki de projeto no GitLab. Abaixo segue os templates para cada documento:

- Plano de Arquitetura
- Plano de Infraestrutura

O PLA deve ser o documento inicial a receber aprovação, por meio da abertura de um chamado para a SECTI/COARQ. Posteriormente, um chamado deve ser aberto para a GPROD/COBAN para a aprovação do PLI e a criação do ambiente de deployment.

## Documentação Geral e Wiki

A página *Home* da Wiki do projeto deve conter as seguintes informações:

- Breve descrição
- Responsável Técnico
- Responsável de Negócio
- Características do projeto, conforme documentado neste link

## Configurações do GitLab

### Grupos de Sistemas no GitLab

O grupo Sistemas Bancarios é responsável por agrupar todos os projetos de software. Internamente, possui um primeiro nível de sub-grupos que reunem projetos de mesma área, que são:

- Crédito
- Automação Bancária
- Core Bancário
- Administrativo
- Integrações

De acordo com a arquitetura corporativa, vários tipos de componentes de software podem ser construídos para um determinado projeto. Por exemplo, um projeto chamado *Crédito Fomento* pode ser composto por APIs, interfaces gráfica para web, apps para mobile e jobs diários.

Para fins de organização destes componentes de software, deve-se criar um **Grupo Geral** relacionado à solução como um todo e incluir neste os repositórios relacionados. A estrutura final é exemplificada abaixo*:

```
├── Sistemas Bancários
|   ├── Crédito
|   |   ├── Controper
|   |   |   ├── api-credito-fomento
|   |   |   ├── credito-fomento
|   |   |   ├── mobile-credito-fomento
|   |   |   ├── job-fomento-processamento-diario
|   ├── Automação Bancária
|   ├── Core Bancário
|   ├── Administrativo
|   ├── Integrações
```

### Configuração de username e e-mail:

Para registro correto de commits, é necessário configurar o **username** e **e-mail**, conforme registrado no LDAP do Banco da Amazônia. O comando abaixo realiza esta configuração de forma global:

bash
```
git config --global user.name "Fulano de Tal"
git config --global user.email "fulano@provedor.com.br"
```

### Quebra de Linha Padrão

Essas configurações não são obrigatórias, pois a instalação já as aplica corretamente. No entanto, estão descritas aqui para o caso de o desenvolvedor precisar ajustar um repositório que foi transferido de um sistema operacional para outro.

Windows Linux

bash
```
git config --global core.autocrlf true
```

### Incluindo Certificados Digitais

Utilize os comandos abaixo, de acordo com o seu sistema operacional, para configurar os certificados digitais no Git.

Windows Linux/Debian

bash
```
# Necessita que o Git Credential Manager esteja instalado
# e os certificados estejam configurados Trust Store do Windows
git config --global http.sslBackend schannel
```

### Credential Manager (Windows)

Ao instalar o Git Client no Windows, temos acesso Git Credential Manager para o armazenamento seguro de senhas. Utilize o comando abaixo para linkar todos os repositórios Git ao Credential Manager.

Windows WSL2/Debian

bash
```
git config --global credential.helper manager-core
```

## Instalação de Certificados

Abaixo, seguem as instruções para instalação de certificados em sistemas operacionais Windows e Linux.

### Instalação de Certificados no Windows

Para instalar os certificados no Windows 10:

1. Clique duas vezes para executar o arquivo e pressione o botão "*Instalar Certificado*".
2. Em "*Local de Armazenamento*", selecione "*Máquina Local*". Pressione Avançar.
3. Certifique-se de que a opção "*Colocar todos os certificados no repositório a seguir*" esteja selecionada. Pressione Procurar.
4. Na caixa de diálogo "*Selecionar Repositório de Certificados*", marque Mostrar repositórios físicos e selecione *Autoridades de Certificação Raiz Confiáveis > Computador local*.

### Instalação de Certificados no Ubuntu/Debian

Para instalar os certificados no Ubuntu/Debian:

> Observação: É crucial que o arquivo tenha a extensão .crt, pois sem ela, o arquivo não será processado.

1. Copie os arquivos de certificado para a pasta */usr/local/share/ca-certificates*.
2. Execute o comando abaixo para atualizar o repositório de certificados:

bash
```
# Instala o pacote de certificados caso não esteja instalado
sudo apt-get install -y ca-certificates
# Atualiza o repositório de certificados
sudo update-ca-certificates
```



Created by MARYONE MAIA DIAS

https://manual-desenvolvimento-software.ppps.ocp-prd.hanceoamazonia.sa



**Arquitetura Corporativa**
Introdução
Padrões Arquiteturais
Políticas de Segurança
Configuração e Governança
Fluxo CI/CD
APIs REST

**Padrões de Desenvolvimento**
Java
Web
Nomenclatura para Banco de Dados
Configuração OpenShift

# Padrões de CI/CD

O Continuous Integration/Continuous Deployment (CI/CD) é uma prática fundamental na engenharia de software, visando automatizar e agilizar o processo de desenvolvimento, teste e entrega de software. Nesta documentação são apresentados os padrões de CI/CD adotados no Banco da Amazônia.

## Visão Geral do Fluxo de CI/CD

Na nossa instituição, a implementação de um pipeline de desenvolvimento eficiente é crucial para simplificar a gestão de configuração e fortalecer a segurança dos sistemas. Nesse sentido, adotamos o GitLab CI/CD como ferramenta central para automatizar e orquestrar nossos processos de análise, build e deployment. Integrando o GitLab a esse pipeline, alcançamos uma infraestrutura altamente portátil e consistente para a execução de tarefas em conformidade.

O Docker assegura a replicabilidade e consistência do ambiente de execução, independentemente da infraestrutura subjacente, linguagem ou plataformas de programação. Com containêineres, encapsulamos componentes essenciais para análise, build e deployment, garantindo a confiabilidade das aplicações em qualquer ambiente, mantendo a flexibilidade para adaptação às novas tecnologias e requisitos.

Neste grupo de repositórios do GitLab, estão localizados projetos contendo configuração de imagens Docker para execução de CI/CD, que incluem scripts para ferramentas como Maven, NodeJS, OpenShift CLI, SonarQube, Database Migrations, entre outros. Por questões de segurança, o acesso a este grupo é restrito a membros da equipe de infraestrutura e arquitetura.

> **Dica**
>
> Neste artigo você pode encontrar um guia para iniciantes sobre GitLab CI/CD.

As seguintes ferramentas compõem a implementação do CI/CD:

- **GitLab**: Plataforma central para controle de versão, colaboração e gestão de projetos de software.
- **SonarQube**: Ferramenta para análise de qualidade e segurança de código.
- **Docker**: Para execução de tarefas de análise, build e deployment em Containêineres, garantindo a portabilidade e consistência do ambiente de execução.
- **OpenShift**: Plataforma de containêineres baseada em Kubernetes, onde as aplicações são implantadas.
- **Artifactory e Nexus**: Repositórios de artefatos para gerenciamento de pacotes, dependências e artefatos de build.

## Implementação do Pipeline

É no projeto Pipeline Templates onde são construídas módulos de configuração para cada tipo de projeto, como Java, VueJS, bem como a estrutura geral do processamento do pipeline.

Há três principais módulos de configuração implementados:

- **Módulo de workflow**: compõem múltiplos fluxo de trabalho para diferentes plataformas de implantação de sistemas, sendo os principais o OpenShift 3 e OpenShift 4, onde são definidos nomes de objetos de deployment e configurações relativas ao ambiente de implantação (desenvolvimento, homologação e produção).
- **Módulo para Vue.js**: configura o pipeline para projetos VueJS, onde são definidas as tarefas de construção e artefatos específicos referentes a este tipo de tecnologia.
- **Módulo Java/Maven**: configura o pipeline para projetos Java que utilizam Apache Maven como gerenciador de dependências e build.

A medida que o Banco da Amazônia adota novas tecnologias e frameworks, novos módulos de configuração podem ser criados.

### Estágios do GitLab CI/CD

O pipeline de CI/CD é composto por cinco estágios principais para organização de tarefas executadas:

- *initialize*: Inicialização do ambiente para construção e deployment do projeto
- *build*: Construção dos artefatos de build para o projeto
- *qa*: Análises de qualidade e segurança
- *deploy*: Disponibilização do build no ambiente de deployment relacionado
- *finalize*: Geração de evidências e notificações relacionadas à execução do pipeline

Os estágios *initialize* e *finalize* são utilizados internamente para tarefas comuns a todos os projetos, independente de sua arquitetura. Os desenvolvedores utilizarão mais frequentemente os estágios de build, qa e deploy para executar tarefas de CI/CD, orientados pela arquitetura dos sistema.

## Configuração do Pipeline

Para configurar os projetos para execução do pipeline, é necessário incluir o arquivo `.gitlab-ci.yml` na raiz do repositório do projeto. Este arquivo contém as instruções para execução das tarefas executadas do pipeline. Segue um exemplo de arquivo de configuração para um projeto VueJS:

▸ **Exemplos de Configuração do GitLab CI/CD**

O pipeline é construído de forma componentizada, onde cada projeto importa módulos de configuração específicos para sua linguagem e tecnologia. Isso permite a reutilização de configurações comuns e a customização de tarefas específicas de cada projeto.

Do exemplo fornecido, o código importa o módulo `.gitlab-ci.yaml` do pipeline **Pipeline Templates** de onde são importadas as configurações comuns e indica a inclusão dos módulos de Workflow (fluxo de trabalho) e de configuração específica para o projeto VueJS:

```yaml
include:
  - project: $PIPELINE_TEMPLATE_PROJECT_PATH
    ref: master
    file:
      - "$PIPELINE_WORKFLOW"
      - "/templates/vuejs-template.gitlab-ci.yml"
```

> **Dica**
>
> O uso de variáveis de ambiente em arquivos de configuração (como `$PIPELINE_WORKFLOW`) facilita mudanças de configuração para múltiplos projetos sem necessidade de edição direta do arquivo `.gitlab-ci.yml`, no controle de versão.

Os arquivos de configuração do GitLab CI/CD para cada tipo de projeto estão detalhados em capítulos próprios nesta documentação.

### Variáveis de Configuração

Para facilitar a configuração dos pipelines, são utilizadas variáveis de CI/CD definidas por projeto e podem ser utilizadas em qualquer parte do arquivo de configuração. As principais variáveis para conhecimento são:

> O menu de variáveis de CI/CD é acessado através do menu do projeto no GitLab, em Settings > CI/CD > Variáveis.

- `PIPELINE_WORKFLOW`: Caminho para o módulo de workflow do projeto, de acordo com as características do sistema. Não precisa ser alterado pelo desenvolvedor.
- `PL_IMAGE`: Imagem Docker utilizada para levantar o sistema no OpenShift.
- `PL_PROJECT_TYPE`: Tipo de projeto, como java, vuejs, jboss, entre outros.

Abaixo segue a lista de imagens disponíveis para a variável `PL_IMAGE`:

- basa-custom-eap71:1.0
- basa-custom-eap71hrcalculo
- basa-custom-eap72:1.0
- basa-custom-eap73:latest
- redhat-openjdk18-openshift:1.5
- openjdk-11-rhel7:latest
- openjdk-17:latest
- openjdk-21:latest
- nginx-116-rhel7:latest
- nginx-120-rhel8:1.58

## Gestão de Branches

Em conformidade com o princípio de separação de ambientes, são criados namespaces no OpenShift para cada ambiente de deployment. Utilizamos um padrão de nomenclatura de branches para representar cada ambiente, como descrito a seguir:

| Ambiente | Branch | Descrição |
|---|---|---|
| Desenvolvimento | `feature` | Utilizada para desenvolvimento ativo do projeto, representa uma funcionalidade para ser testada em ambiente de desenvolvimento. |
| Homologação | `release` | Branches para disponibilização de sistemas para homologação. |
| Produção | `master` | Branch padrão do Git, a qual é atualizada quando uma aplicação deve ser disponibilizada em produção. |

Ao enviar alterações de código-fonte para o GitLab, ele detecta estas mudanças e dispara o pipeline automaticamente de acordo com a branch. Ao final, o pipeline fará o deploy no ambiente correspondente.

Em relação à branch de produção, é importante ressaltar que ela deve ser atualizada apenas através de um merge request homologado pela equipe técnica e gestora. Qualquer alterações realizadas diretamente nesta branch inicia o processo de deployment, por isto o nível de acesso necessário para alterações nesta branch deve ser restrito.

### Múltiplas Equipes e Deployments

Para acomodar várias equipes de desenvolvimento no funcionalidades, em vez de um único deployment por ambiente, podem ser implantadas múltiplas versões da aplicação a partir de **branches de deployment especializadas** para cada ambiente.

Estas branches podem ser geradas utilizando seu nome padrão como prefixo, seguido de um texto descritivo, separados por hífen.

**Exemplos de Branches Especializadas**

- `feature-issue-01` → Branch para desenvolvimento de uma funcionalidade específica.
- `release-funcionalidade-agencia` → Branch para homologação de uma funcionalidade específica para agências.

> **Aviso**
>
> Evite criar branches especializadas para cada desenvolvedor ou tarefa, pois consomem recursos e licenças. Crie novos ambientes somente quando necessário e temporariamente.

### Configurações Especializadas no OpenShift

Cada namespace no OpenShift é configurado para receber as aplicações de acordo com a branch correspondente. Cada um possui um volume ConfigMaps e Secret montado para configuração da aplicação, compartilhado entre todas as unidades de deployment daquele namespace.

Mas há casos onde a aplicação de uma branch especializada possua necessidades diferentes da configuração geral, com exemplo, acesso a um banco de dados específico.

Nestes casos, configure `configMaps` e/ou `secrets` para cada branch especializada, nomeando-os com o mesmo nome da branch com o sufixo correspondente ao objeto a ser criado.

**Exemplo de ConfigMap e Secret Especializados**

- `feature-issue-01-configmap` → ConfigMap para a branch `feature-issue-01`.
- `release-funcionalidade-agencia-secret` → Secret para a branch `release-funcionalidade-agencia`.

## Análise de Qualidade

A análise de qualidade e segurança é realizada pelo SonarQube, que é integrado ao processo de CI/CD para execução de análises estáticas de código-fonte para todas as aplicações desenvolvidas no Banco da Amazônia.

Todos os usuários de rede do Banco da Amazônia possuem acesso ao SonarQube, através deste link, não havendo necessidade de solicitações de acesso.

> **Dica**
>
> A inclusão dos projetos no SonarQube para análise é feita automaticamente após a primeira análise realizada

Cada sistema analisado gera três registros no SonarQube para os ambientes de desenvolvimento, homologação e produção. A chave é o nome são derivados do nome do sistema e do sufixo do ambiente (dev, hml, prd). Assim, um sistema chamado *api-sistema* terá três registros:

- **api-sistema-dev**
- **api-sistema-hml**
- **api-sistema-prd**

O CI/CD aguarda feedback do Sonar sobre a análise do projeto. Se detectar problemas de qualidade ou segurança, uma exceção é lançada com uma mensagem e um link para os detalhes da análise. Se essa mensagem aparecer, acesse o link para ver os detalhes e corrigir os problemas.

**Exemplo de Mensagem de Erro**

```log
Caused by: org.sonar.api.utils.MessageException: Quality gate status: FAILED - View details on https://10.3.21.102:9000/dashboard?id=sonarqube-example
```

### Quality Gates

Quality Gate é um recurso do SonarQube que define o critério de aceitação de qualidade do código. É essencialmente um conjunto de condições que o código deve atender antes de ser considerado apto para produção. O código não passar pelo Quality Gate, o SonarQube sinalizará o projeto como falho. As condições podem incluir itens como cobertura de testes, complexidade ciclomática, dívida técnica, etc. Dois Quality Gates foram criados como padrão, *Base* e *Legado*.

Este recurso possui duas configurações principais, **Análise Geral (Overall Code)** e **Análise de Novos Códigos (New Code)**, onde diferentes regras de análise podem ser aplicadas. "Overall Code" refere-se a todo o código do projeto, enquanto "New Code" refere-se apenas ao código recentemente adicionado ou modificado.

> **Dica**
>
> Para mais detalhes sobre a configuração do Quality Gate, acesse a documentação oficial.

#### Quality Gate: *Base*

Quality Gate padrão, adotado para todos os novos projetos. Sua configuração geral é demonstrada abaixo:

**Análise Geral (Overall Code)**

- *Percentual de Linhas Duplicadas*: maior que 30%
- *Rating de Manutenibilidade*: pior que A
- *Quantidade de Code Smells*: maior que 2.500
- *Rating de Confiabilidade (Reliability)*: pior que B
- *Rating de Segurança*: pior que A

**Análise de Novos Códigos (New Code)**

- *Test Coverage*: menor que 80%
- *Percentual de Linhas Duplicadas*: maior que 30%
- *Rating de Manutenibilidade*: pior que A
- *Quantidade de Code Smells*: maior que 2.500
- *Rating de Confiabilidade (Reliability)*: pior que B
- *Rating de Segurança*: pior que A

#### Quality Gate: *Legado*

Utilizado para aplicações legadas que já possuem um grande débito técnico acumulado, possuindo regras mais brandas. Ignora todos os bugs e débitos técnicos existentes, por isto não há regras para Overall Code, mas novos problemas devem ser corrigidos, como demonstrado abaixo:

Este Quality Gate deve ser configurado manualmente por projeto.

**Análise de Novos Códigos**

- *Percentual de Linhas Duplicadas*: maior que 40%
- *Rating de Manutenibilidade*: pior que A
- *Rating de Confiabilidade (Reliability)*: pior que B
- *Rating de Segurança*: pior que A

## Registro de Indicadores

Dados importantes do fluxo de CI/CD são registrados para monitoramento e auditoria. A seguir estão listados os principais indicadores.

### Ferramenta de APM

As ferramentas de APM (Application Performance Management) são usadas para monitorar e gerenciar o desempenho de aplicações de software. Elas fornecem insights em tempo real sobre a performance de aplicações, ajudando a identificar e corrigir problemas rapidamente para garantir uma experiência de usuário otimizada.

O Dynatrace é a ferramenta de APM padrão do Basa, a qual recebe do pipeline de CI/CD notificações de cada deploy em homologação e produção para criar marcos no monitoramento das aplicações.

### Tags em Commits do Git

Ao realizar deploys para ambiente produtivo, o commit utilizado para realização da build é associado a uma tag contendo informações:

- Namespace alvo no OpenShift
- Versão da Aplicação

O nome da tag é composto pela versão do sistema e id do pipeline no formato `<VERSAO_DO_PROJETO>#<ID_DO_PIPELINE>`, por exemplo `v1.0.1#42`.



Made with MATERIAL FOR MKDOCS

On this page
Visão Geral do Fluxo de CI/CD
Implementação do Pipeline
Configuração do Pipeline
Gestão de Branches
Análise de Qualidade
Registro de Indicadores

# Padrões para APIs REST

*REST (Representational State Transfer)* é uma arquitetura onde dados e funcionalidades são considerados recursos, acessados via URIs padronizadas. Os recursos são manipulados através de operações simples e sem estado. A definição de recursos reais e operações específicas para cada caso de uso será identificada e mencionada no artefato.

## Documentação das APIs

A documentação das APIs é feita por meio da ferramenta OpenAPI/Swagger. Ela fornece informações detalhadas sobre como usar a interface com as APIs, incluindo endpoints, métodos, parâmetros, formatos de solicitação e resposta e códigos de status de resposta e muito mais. A OpenAPI Specification (OAS) é, antes, baseada em Especificação v2 e pode gerar documentação separada de acordo com as APIs em JSON, substituindo documentação manual. Cada API tem seu próprio URL que, conforme a OAS, contém uma descrição clara de suas operações. Aplicações podem usar interfaces públicas para simplificar a leitura da documentação. Clientes podem entender como interagir com a API.

## Padrão para Criação de URIs

As URIs são estruturadas dessa forma para ser sequencial e intuitiva: `/topico / id / (versão) / (recurso)` por exemplo, `/divulgacao/relatorios/1`. Abaixo segue a descrição de cada elemento:

- **Host:** endereço base da API. Ex.: `https://api.sistema.NOME.bancoamazonia.co`;
- **"api":** prefixo usado para identificar APIs;
- **Versão:** versão da API. Inicia-se em "v1" e aumenta-se apenas em caso de mudanças significativas;
- **Recurso:** O recurso a ser consumido.

Nomes de recursos devem ser escritos no plural e em kebab-case, sendo verbos e uso de caracteres especiais, a menos caracteres em caixa alta ou underscore (_).

Evite uso de adjetivos como nome para criação de recursos (Ex.: `POST /home-recurso`), pois o método HTTP já indica a ação que será executada.

Como regra geral:

1.Não utilize o caractere "/" ao final das URLs

2.Evite o uso de extensões de arquivos para definir o formato esperado de recursos, como `.xml` ou `.json`. Em vez disso, utilize o cabeçalho `Accept` da requisição e o cabeçalho HTTP header `Accept` com o `MIME type` correspondente para definir o formato de retorno do recurso.

3.Evite incluir a definição das URIs, incluindo para retornar ao localização do recurso, mantenha a URI simples e redirecione o cliente para o novo local/ação, utilizando o código HTTP 301.

4.Em caso de dúvida sobre o uso de atributos ou hierarquia, considere:

- Se existe uma relação de hierarquia ou de pertencimento do recurso, utilize a Request Parameter
- Se não existe relação de hierarquia ou de pertencimento do recurso, como dados de autenticação, utilize o Header.

## Estrutura de Endpoints

Abaixo segue os padrões de definição de URLs para cada tipo de operação em recursos da API:

| Operação | Método HTTP | URL | Objetivo da Solicitação |
|---|---|---|---|
| Criação | POST | `/recurso` | 201 com representação recurso |
| Listagem | GET | `/recurso` | Lista paginada de DTOs que representam o recurso |
| Consulta | GET | `/recurso/{ID}` | 200 com representação recurso |
| Atualização Completa | PUT | `/recurso/{ID}` | 200 com representação recurso |
| Atualização Parcial | PATCH | `/recurso/{ID}` | 200 com representação recurso |
| Deleção | DELETE | `/recurso/{ID}` | Sem retorno |

Todos pesquisa no Swagger do recurso de request e parâmetros e filtrada da query parameters, como exemplo: `/recurso?campo=valor&page=0&size=10`.

## Diferença Entre PUT e PATCH

O HTTP PUT e PATCH são métodos usados para atualizar recursos.

PUT substitui todo o recurso atual pelo novo recurso. Se certos atributos não estão presentes no novo recurso, PUT os removerá.

PATCH, por outro lado, aplica modificações parciais aos atributos especificados, deixando o resto do recurso inalterado. É útil quando você quer alterar apenas algumas partes de um recurso.

## Formato de Comunicação

As APIs REST devem usar JSON para comunicação. Amplo, devido à sua leveza e ampla compatibilidade. Ao enviarmos dessa maneira segue o padrão HTTP, em código de status e mensagens claros, e, quando aplicável, o corpo da resposta deve conter o objeto de retorno.

**Aviso**

Evite o uso de entidades de domínio como objetos de retorno, pois isso pode expor informações sensíveis não desnecessárias para o cliente.

## Códigos de Status HTTP

Os códigos de status HTTP são responsáveis padronizados que indicam o resultado de uma solicitação HTTP.

A tabela a seguir lista os códigos de status mais comuns e seus significados por método HTTP:

| Método HTTP | Status Response Code | Objetivo |
|---|---|---|
| GET | 200 OK | Consulta de recursos |
| POST | 201 Created | Criação de novos recursos |
| POST | 202 Accepted | Início de operações assíncronas |
| PUT | 200 OK | Atualização completa de recursos |
| PATCH | 200 OK | Atualização parcial de recursos |
| DELETE | 204 No Content | Deleção física ou lógica de recursos |

Os códigos de status HTTP a serem utilizados como respostas de erro são:

- **400 Bad Request:** requisição malformada pelo cliente, como dados ou estrutura inválidos
- **401 Unauthorized:** dados de autenticação ausente ou inválido
- **403 Forbidden:** usuário não possui acesso ao recurso ou uma política de segurança foi violada
- **404 Not Found:** recurso solicitado não existe
- **405 Method Not Allowed:** método HTTP não permitido para o recurso
- **422 Unprocessable Entity:** requisição bem formada, mas com dados inválidos
- **500 Internal Server Error:** erros inesperados no servidor, como falhas de infraestrutura ou de configuração

## Headers HTTP

Os headers HTTP são informações adicionais enviadas com a requisição HTTP. Os headers recomendados para requisições de API REST são:

- **Content-Type:** define o tipo de mídia do recurso. Deve ser `application/json` para requisições de JSON.

## Data Transfer Objects

Os Data Transfer Objects (DTOs) são usados nas APIs REST para serem encapsulados os dados que serão enviados e recebidos. Eles garantem que a API retorna os formatos consistentes de dados, facilitam a validação e ajudam a ocultar detalhes de implementação do cliente.

Devem ser construídos com validações para garantir a integridade dos dados e com métodos `fromNewResource`, `fromCheckResource`, `fromCheckResource`, `fromCheckResource` e `fromCheckResource` (`RequestNewResource`).

O uso de DTOs contribui para a segurança, manutenibilidade e eficiência das APIs REST, sendo uma prática recomendada por vários motivos:

- **Controle sobre os dados:** DTOs permitem que você controle exatamente quais dados serão expostos pela sua API. Você pode selecionar apenas os campos relevantes para a resposta, evitando o excesso de informações que podem ser desnecessários ou sensíveis.
- **Evitar exposição excessiva:** Expor diretamente suas entidades de negócio pode criar um acoplamento desnecessário entre o modelo de dados interno da sua aplicação e a estrutura da sua API. Isso pode tornar mais difícil fazer mudanças em um dos lados sem afetar o outro.
- **Segurança:** Ao utilizar DTOs, você pode filtrar e formatar os dados para garantir que informações sensíveis não sejam inadvertidamente expostas através da sua API. Isso ajuda a manter a segurança dos dados do seu sistema.
- **Versionamento simplificado:** Se você precisar fazer alterações na estrutura de dados retornada pela sua API, pode ser mais fácil introduzir essas mudanças em DTOs sem afetar diretamente as entidades de negócio. Isso simplifica o processo de versionamento da API.
- **Desempenho:** Em muitos casos, as entidades de negócio contêm informações adicionais ou relacionamentos complexos que não são necessários para a resposta da API. Usar DTOs permite que você otimize a estruturação dos dados enviados, o que pode melhorar o desempenho da API ao reduzir o tamanho das respostas.
- **Flexibilidade:** DTOs fornecem flexibilidade na estruturação dos dados retornados pela sua API. Você pode organizar os DTOs de acordo com as necessidades específicas de diferentes endpoints da sua API, tornando mais fácil para os clientes consumirem os dados de maneira eficiente.

## Formato para Erros

Os clientes de uso em uma API REST são executados para comunicar problemas ocorridos durante o processamento de uma solicitação. Eles fornecem detalhes sobre o erro, como o código de status, a mensagem de erro e, possivelmente, informações adicionais, ajudando o cliente a entender e resolver o problema. Adotamos o padrão RFC 9457 para representação de erros em APIs REST.

**Informações Extras**

A RFC 9457 também detalha Ex.HTTP HTTP, define um formato para transportar detalhes de erros em respostas HTTP. Ele inclui informações como o código de status HTTP, uma descrição do erro e, opcionalmente, informações adicionais específicas do contexto. Isso ajuda os clientes a entender e lidar com erros de forma mais consistente e eficiente.

Abaixo segue um exemplo de objeto de erro:

```
{
    "type": "about:blank",
    "title": "Bad Request",
    "status": 400,
    "detail": "Erro!",
    "instance": "/api/v1/entidade/123",
}
```

- `type`: Um URI que identifica o tipo de problema. Pode ser `about:blank` para erros genéricos.
- `status`: O código de status HTTP gerado no servidor para a ocorrência do erro.
- `title`: Uma sumário curto e humanamente legível do problema.
- `detail`: Uma explicação detalhada específica para esta ocorrência do problema. Mais informações sobre os campos do que apresentam informações específicas.
- `instance`: Uma URI que identifica a entidade específica que ocorreu o problema.

## Query Parameters

Os Query Parameters em APIs REST são usados para filtrar, ordenar, ou controlar a resposta de API. Eles são adicionados à URL após um "?" e são separados por "&". Por exemplo: `/api/usuarios?ordenacao=nome&limite=10`. `sort` e `limit` são Query Parameters.

Seus nomes devem ser autoexplicativos, sem redundância de entendimento e escritos no singular segundo o padrão Camel Case, por exemplo, `/usuarios?nome=Joao`.

Adicione qualificadores complementares quando necessário para refinar o significado, por exemplo, `dataInicio` e `dataFim`.

- **nomes:** Ex: nomeCliente
- **datas:** Ex: dataCadastro ou dataUltimoAcesso
- **horários:** Ex: horaInformacaoAgencia
- **quantidades:** Ex: quantidadeRegistros
- **textos explicativos:** Ex: descricaoPropriedade

Métodos booleanos devem ser formulados usando perguntas preferiveis com verbos, por exemplo, `possuiAutorizacaoAtivo`.

## Paginação

A paginação é um recurso importante na API REST que retornam grandes volumes de dados. Ela permite que os clientes solicitem apenas uma parte dos dados de cada vez, reduzindo o tempo de resposta e o uso de recursos do servidor, além de evitar atrasos de rede e sobrecarga do cliente. Existem duas formas principais de fazer isso, mas estamos padronizando o filtro de paginação com **`springframework`**.

As APIs não precisam gerenciar totalmente de manutenção na paginação. Portanto, os dados retornados podem variar entre requisições de página devido a alterações nos dados subjacentes.

Os parâmetros de paginação devem ser passados na Query Parameters e devem seguir o seguinte padrão:

- **page:** número da página a ser retornada. O número padrão dos páginas inicia zero-zero (0) se não for fornecido.
- **size:** quantidade de registros por página. O valor padrão é 20 e o valor máximo é 500, sendo o valor de ordenação dos registros.

A ordenação pode incluir a direção do resultado, como `sort=nome,asc` ou `sort=nome,desc`, para ascendente ou descendente respectivamente. Caso a ordenação não seja especificada, assume ordenação ascendente.

É esperado que a API retorne os totais de tamanho de página e número de páginas, além de `totalElements` e `totalPages`, para o cliente. Isso é importante para navegação.

Abaixo segue a Json Schema referente ao padrão adotado para paginação:

```
{
  "data": [],
  "links": {
    "first": "string",
    "prev": "string",
    "next": "string",
    "last": "string"
  },
  "meta": {
    "totalRecords": 0,
    "totalPages": 0,
    "pageNumber": 0,
    "pageSize": 0
  }
}
```

O atributo `data` representa a lista de resultados em si, ou seja, a página atual. O atributo `links` representa a forma de navegação entre as páginas, respectivamente filtrados de consulta adicionais e possuem os seguintes atributos:

- `last`: URL para a página corrente. Obrigatório.
- `first`: URL para a primeira página. Obrigatório se a resposta não for a primeira página.
- `prev`: URL para a página anterior. Obrigatório se a resposta não for a primeira página.
- `next`: URL para a próxima página. Obrigatório se a resposta não for a última página.
- `last`: URL para a última página. Obrigatório se a resposta não for a última página.

O atributo `meta` define informações sobre a paginação, como:

- `totalRecords`: O número total de registros encontrados
- `totalPages`: O número total de páginas
- `pageSize`: O número de registros por página
- `currentPage`: O número da página atual

## Validação e Sanitarização dos Dados de Entrada

É boa prática verificar se os dados enviados para uma determinada operação estão em formato e quantidade adequados. Nos deveremos apoiar em bibliotecas prontas, evitando de realizar uma regulação, levando a uma manutenção custosa e propensa a erros. Também devemos garantir que os atributos enviados estejam de acordo com os critérios de negócio.

No contexto da especificação JPA, onde usar anotações de domínio como parâmetros de entrada para operações de atualização. Em vez disso, use DTOs para representar os dados de entrada e converta-os para entidades de domínio, se necessário.

## Autenticação e Autorização

A autenticação e autorização são importantes funcionalidades de segurança de uma API. A autenticação é o processo de verificar a identidade de um usuário, enquanto a autorização é o processo de verificar se um usuário autenticado pode acessar um recurso.

Veja mais detalhes em Política de Segurança

## Obtenção do Acesso Token

O Access Token, também conhecido como Bearer Token, é um token de acesso, o dados para RFC 6750 Bearer Token Usage. Ele é utilizado para as fluxos de autorização OAuth 2.0, como Token 1 e em Bearer Token, que deve ser enviado no cabeçalho `Authorization` de todas as requisições.

Exemplo de cabeçalho de requisição com Access Token: `Bearer myTokenId`

```
GET /api/recursos/exemplo HTTP/1.1
Host: api.sistema.NOME.bancoamazonia.co
Authorization: Bearer myTokenId
```

Para obtenção do Access Token, o cliente deve enviar uma requisição para o endpoint de autenticação do NOME Gateway e as credenciais de acesso e o Client ID de aplicação. O Client ID é um identificador único da aplicação, fornecido pelo time de arquitetura.

O "Client Secret" é uma senha compartilhada entre a aplicação e o IAM SSO, que deve ser mantida em segredo.

Para ambientes de desenvolvimento e homologação, o Client ID e Client Secret podem ser obtidos com a equipe de DTM-D.

Abaixo segue exemplos de requisições para obtenção do Access Token em ambiente de homologação, utilizando o IAM SSO do Banco da Amazônia para os fluxos de autorização Client Credentials e Password.

O Client Credentials é utilizado para autenticação de aplicações, como demonstrado abaixo:

**Linux / Windows**

```
curl -X --request POST \
  --url https://login-hml.bancoamazonia.co.br/auth/realms/base
  realm/protocol/openid-connect/token \
  --data grant_type=client_credentials \
  --data client_id=api-clientes \
  --data client_secret=4c4b6b22-c178-4f9d-9dc5-3239f5b12c3d
```

O grant type Password é utilizado para autenticação de usuários, como demonstrado abaixo:

**Linux / Windows**

```
curl -X --request POST \
  --url https://login-hml.bancoamazonia.co.br/auth/realms/base
  realm/protocol/openid-connect/token \
  --data grant_type=password \
  --data client_id=api-clientes \
  --data client_secret=4c4b6b22-c178-4f9d-9dc5-3239f5b12c3d
  --data username=usuario \
  --data password=senha
```

A resposta da requisição retorna um objeto JSON cujo atributo `access_token` possui o valor desejado.

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwi...",
  "expires_in": 300,
  "refresh_expires_in": 1800,
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOiAiSldUIiwiaWQiOiJjMTY...",
  "token_type": "Bearer",
  "not-before-policy": 0,
  "session_state": "1c79e57b-1268-48c7-a8e2-d43c1b0ec67c",
  "scope": "email profile roles openid"
}
```

## Script para Renovação de Access Token no Postman

O Postman é uma ferramenta de colaboração para desenvolvimento de API. Abaixo segue um script para renovação de Access Token no Postman.

- Cliques para ver o script

## Permissões de Acesso

As permissões de acesso servem para regular quais usuários ou conjuntos de usuários possuem autorização para acessar recursos específicos. No nosso Keycloak, esses recursos são representados e gerenciados através de diferentes escopos. Para mais informações, consulte a seção Política de Segurança.

As APIs REST são a responsabilidade de assegurar que o usuário autenticado atenda as permissões requeridas para acessar o recurso ou executar uma operação. Caso o usuário não possua as permissões necessárias, a API deve retornar o código de status HTTP 403 Forbidden.

A API Gateway é responsável por validar o objeto de formatar a fase de tratativa para um usuário tem para um sistema específico. A seguir, apresentamos um exemplo de `curl` para obter o token de acesso para o sistema `SISTEMA` e para listagem para o sistema `api-consulta`.

```
curl --request GET \
  --url 'https://api.bancoamazonia.co.br/autenticacao/v1/usuarios/1111' \
  --header 'Authorization: Bearer {{access_token_sistema}}' \
  --header 'Content-Type: application/json'
```

Para recuperar as transações associadas a uma conta de serviço, como uma API, substitua a matrícula do usuário pelo Client ID na requisição. Por exemplo, a URL da requisição seria `https://api.bancoamazonia.co.br/autenticacao/v1/usuarios/api-consulta`, onde `api-consulta` é o Client ID do sistema que está sendo consultado ao transações para o sistema `api-consulta`.

**Aviso**

O acesso a um recurso via API protegida é permitido somente se o usuário (ou conta de serviço) possuir as mesmas uma transação ou recurso ao recurso questão.

O Ambiente Framework, uma biblioteca criada de ferramentas que facilita o desenvolvimento de APIs REST, disponibilizou mecanismos para configuração e validação de permissões, através de anotações. Para mais informações mais detalhadas, consulte a seção correspondente na documentação do Ambiente Framework.

On this page
Documentação das APIs
Padrão para Criação de URIs
Estrutura de Endpoints
Formato de Comunicação
Formato para Erros
Query Parameters
Paginação
Validação e Sanitarização de...
Autenticação e Autorização



Made with Material for MkDocs
Copyright © 2024 Banco da Amazônia

Manual de Desenvolvimento

Arquitetura Corporativa
Arquitetura
Padrões Arquiteturais
Padrões de Desenvolvimento
Configurações Corporativas
Temas JSTL
Versões Suportadas

Ambiente de Desenvolvimento
Java
Implementações para Banco de Dados
Configurações Específicas

Em Destaque
Versões Suportadas
Padrões de Desenvolvimento Java
Ferramentas de Build
Configuração do CI/CD
Ferramentas de Qualidade
Extensões de Desenvolvimento
Padrões de Implementação
Logging e Auditoria
Certificados Digitais
Eclipse + plugins

# Padrões de Desenvolvimento Java

O Maven pode ser utilizado e integrado com o seu ou com qualquer IDE para gerenciar as dependências, compilar, executar testes unitários e empacotar a aplicação. Ele é recomendado, e é o padrão adotado pelo projeto PJe.

## Versões Suportadas

Atualmente o PJe é compatível com a Java Platform Standard Ed. que é distribuída e suportada pela Oracle nas versões 11 e 17.

As imagens Docker oficiais para execução da aplicação estão disponíveis para ambas as versões, veja mais em Docker Container Images.

Nota

Para executar o PJe no Eclipse, é necessário ao configurar o Servidor para o projeto Primeiro, configurar o JRE para o Java 11 ou 17.

## Ferramentas de Build

O Apache Maven é a ferramenta de build oficial para projetos Java da Oracle do Brasil. Entretanto, é flexível o suficiente para a execução de compilação e gerenciamento de dependências de outras ferramentas como o Gradle, Ant, Ivy, entre outras.

Para a execução básica de build, execute o seguinte comando na raiz do projeto:

```
$ mvn clean install -DskipTests=true -Dmaven.javadoc.skip=true -B -V
```

Para executar testes unitários e de integração, veja a seção Testes Unitários e Testes de Integração.

### Arquivo settings.xml

Caminho de um conteúdo customizável para o seu arquivo settings.xml do Maven.

Nele:

- Definimos o JDK a ser usado na execução dos testes (com JUnit)
- Temos o nome do repositório para baixar as dependências

### Como editar o arquivo de instalação do Eclipse

### Arquivo "Standard Maven"

# Configuração do CI/CD

Para manter um padrão, todos os fluxos de CI/CD do Jenkins(R) criam ou se valem de um pipeline com o nome padrão **pje-<sistema>-<branch>**.

### Arquivo pipeline.groovy

O job build-apps realiza a execução de todos os testes de sistema. Ele depende dos parâmetros de JDK, além

- **JDK_SELENIUM_DOCKER_IMAGE**(**MANDATORIO**) Default: **JDK_SELENIUM_PJE_HL7**
- **JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO** Default: **JDK_SELENIUM_PJE_HL7**
- **JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL** Default: **JDK_SELENIUM_PJE_HL7**

O job deploy-apps realiza o deploy da aplicação em ambiente pré-definido, por padrão.

Existe dois jobs padrão de pipeline para o build e deploy de aplicações. Os jobs são configurados para executar o pipeline de mesmo nome, ou seja, o job build-apps executa o pipeline build-apps.groovy e o job deploy-apps executa o pipeline deploy-apps.groovy.

## Versões de JDK para Análise de Qualidade

Alguns projetos legados utilizam versões mais antigas do JDK, como o JDK 6 para o sistema de Processo Administrativo. Para que a análise de qualidade seja executada corretamente, é preciso que o job **pje-build-apps** seja executado com o parâmetro **JDK_SELENIUM_DOCKER_IMAGE** definido como **JDK_SELENIUM_PJE_HL7**.

### Arquivo .gitlab-ci.yml

O job build-apps realiza a execução de todos os testes de sistema. Ele depende dos parâmetros de JDK, além

- **JDK_SELENIUM_DOCKER_IMAGE**(**MANDATORIO**) Default: **JDK_SELENIUM_PJE_HL7**
- **JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO** Default: **JDK_SELENIUM_PJE_HL7**
- **JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL** Default: **JDK_SELENIUM_PJE_HL7**

O job deploy-apps realiza o deploy da aplicação em ambiente pré-definido, por padrão.

Existe dois jobs padrão de pipeline para o build e deploy de aplicações. Os jobs são configurados para executar o pipeline de mesmo nome, ou seja, o job build-apps executa o pipeline build-apps.groovy e o job deploy-apps executa o pipeline deploy-apps.groovy.

## Configuração de Cobertura de Testes

Para obter a cobertura de testes de sistema e de testes no SQLite (H2), adicione o parâmetro **JDK_SELENIUM_DOCKER_IMAGE** com o valor **JDK_SELENIUM_PJE_HL7**.

```
JDK_SELENIUM_DOCKER_IMAGE=JDK_SELENIUM_PJE_HL7
```

Para obter a cobertura de testes de sistema e de testes no PostgreSQL, adicione o parâmetro **JDK_SELENIUM_DOCKER_IMAGE** com o valor **JDK_SELENIUM_PJE_HL7_POSTGRESQL**.

```
JDK_SELENIUM_DOCKER_IMAGE=JDK_SELENIUM_PJE_HL7_POSTGRESQL
```

### Stage cobertura

O relatório de cobertura de testes é produzido automaticamente pelo pipeline, podendo ser acessado no Gitlab através do menu Pipelines e na aplicação no Jacoco.

## Deploy no Bitbucket

Para realizar o deploy no **bitbucket** também é preciso configurar o processo de criação de pipelines. Para isso, crie o arquivo **bitbucket-pipelines.yml** na raiz do seu projeto. Nesse arquivo, insira o conteúdo a seguir. Esse arquivo será utilizado pelo Bitbucket para criar os pipelines de build e deploy.

Na linha 3 e 4 do arquivo, as variáveis **$BITBUCKET_USER** e **$BITBUCKET_PASSWORD** devem ser configuradas no Bitbucket. Para isso, acesse as configurações do repositório e crie as variáveis com os valores de usuário e senha.

Na linha 5 e 6, as variáveis **$GITLAB_USER** e **$GITLAB_PASSWORD** devem ser configuradas no Gitlab. Para isso, acesse as configurações do repositório e crie as variáveis com os valores de usuário e senha.

Na linha 7 e 8, as variáveis **$GITLAB_PROJECT_ID** e **$GITLAB_PIPELINE_ID** devem ser configuradas no Gitlab. Para isso, acesse as configurações do repositório e crie as variáveis com os valores de id do projeto e id do pipeline.

Na linha 9, a variável **$JDK_SELENIUM_DOCKER_IMAGE** deve ser configurada no Gitlab. Para isso, acesse as configurações do repositório e crie a variável com o valor **JDK_SELENIUM_PJE_HL7**.

Na linha 10, a variável **$JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO** deve ser configurada no Gitlab. Para isso, acesse as configurações do repositório e crie a variável com o valor **JDK_SELENIUM_PJE_HL7**.

Na linha 11, a variável **$JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL** deve ser configurada no Gitlab. Para isso, acesse as configurações do repositório e crie a variável com o valor **JDK_SELENIUM_PJE_HL7**.

```
image: atlassian/pipelines-awscli

pipelines:
  default:
    - step:
        name: Build and Deploy
        script:
          - pipe: atlassian/trigger-pipeline:4.1.1
            variables:
              BITBUCKET_USER: $BITBUCKET_USER
              BITBUCKET_PASSWORD: $BITBUCKET_PASSWORD
              GITLAB_USER: $GITLAB_USER
              GITLAB_PASSWORD: $GITLAB_PASSWORD
              GITLAB_PROJECT_ID: $GITLAB_PROJECT_ID
              GITLAB_PIPELINE_ID: $GITLAB_PIPELINE_ID
              JDK_SELENIUM_DOCKER_IMAGE: $JDK_SELENIUM_DOCKER_IMAGE
              JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO: $JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO
              JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL: $JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL
```

### Stage sonar

Nota

Recentemente, foi alterado o nome do repositório Maven padrão do SonarQube. Anteriormente, o nome era sonar-maven-plugin, mas agora foi alterado para sonarsource-sonarqube-mirror. Se você estiver usando uma versão mais antiga do SonarQube, pode ser necessário atualizar a configuração do repositório Maven para refletir essa mudança.

## Configuração de Cobertura de Testes

Para obter a cobertura de testes de sistema e de testes no SQLite (H2), adicione o parâmetro **JDK_SELENIUM_DOCKER_IMAGE** com o valor **JDK_SELENIUM_PJE_HL7**.

```
JDK_SELENIUM_DOCKER_IMAGE=JDK_SELENIUM_PJE_HL7
```

Para obter a cobertura de testes de sistema e de testes no PostgreSQL, adicione o parâmetro **JDK_SELENIUM_DOCKER_IMAGE** com o valor **JDK_SELENIUM_PJE_HL7_POSTGRESQL**.

```
JDK_SELENIUM_DOCKER_IMAGE=JDK_SELENIUM_PJE_HL7_POSTGRESQL
```

### Stage cobertura

O relatório de cobertura de testes é produzido automaticamente pelo pipeline, podendo ser acessado no Gitlab através do menu Pipelines e na aplicação no Jacoco.

## Deploy no Bitbucket

Para realizar o deploy no **bitbucket** também é preciso configurar o processo de criação de pipelines. Para isso, crie o arquivo **bitbucket-pipelines.yml** na raiz do seu projeto. Nesse arquivo, insira o conteúdo a seguir. Esse arquivo será utilizado pelo Bitbucket para criar os pipelines de build e deploy.

Na linha 3 e 4 do arquivo, as variáveis **$BITBUCKET_USER** e **$BITBUCKET_PASSWORD** devem ser configuradas no Bitbucket. Para isso, acesse as configurações do repositório e crie as variáveis com os valores de usuário e senha.

Na linha 5 e 6, as variáveis **$GITLAB_USER** e **$GITLAB_PASSWORD** devem ser configuradas no Gitlab. Para isso, acesse as configurações do repositório e crie as variáveis com os valores de usuário e senha.

Na linha 7 e 8, as variáveis **$GITLAB_PROJECT_ID** e **$GITLAB_PIPELINE_ID** devem ser configuradas no Gitlab. Para isso, acesse as configurações do repositório e crie as variáveis com os valores de id do projeto e id do pipeline.

Na linha 9, a variável **$JDK_SELENIUM_DOCKER_IMAGE** deve ser configurada no Gitlab. Para isso, acesse as configurações do repositório e crie a variável com o valor **JDK_SELENIUM_PJE_HL7**.

Na linha 10, a variável **$JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO** deve ser configurada no Gitlab. Para isso, acesse as configurações do repositório e crie a variável com o valor **JDK_SELENIUM_PJE_HL7**.

Na linha 11, a variável **$JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL** deve ser configurada no Gitlab. Para isso, acesse as configurações do repositório e crie a variável com o valor **JDK_SELENIUM_PJE_HL7**.

```
image: atlassian/pipelines-awscli

pipelines:
  default:
    - step:
        name: Build and Deploy
        script:
          - pipe: atlassian/trigger-pipeline:4.1.1
            variables:
              BITBUCKET_USER: $BITBUCKET_USER
              BITBUCKET_PASSWORD: $BITBUCKET_PASSWORD
              GITLAB_USER: $GITLAB_USER
              GITLAB_PASSWORD: $GITLAB_PASSWORD
              GITLAB_PROJECT_ID: $GITLAB_PROJECT_ID
              GITLAB_PIPELINE_ID: $GITLAB_PIPELINE_ID
              JDK_SELENIUM_DOCKER_IMAGE: $JDK_SELENIUM_DOCKER_IMAGE
              JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO: $JDK_SELENIUM_DOCKER_IMAGE_MANDATORIO
              JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL: $JDK_SELENIUM_DOCKER_IMAGE_OPCIONAL
```

## Principais Dependências Externas

No Banco de Pendências, o uso do Spring Boot com Arquillian Framework é obrigatório para a execução de testes de integração.

O **Spring Boot** é um framework que facilita a configuração e execução de aplicações Java através de convenções pré-definidas. Ele é amplamente utilizado em projetos que buscam simplificar o desenvolvimento de aplicações Java, reduzindo a quantidade de código boilerplate necessário e promovendo uma abordagem mais produtiva e eficiente.

O **Arquillian** é um framework de testes de integração para aplicações Java. Ele permite que os testes sejam executados em um ambiente real, como um servidor de aplicações ou um contêiner, em vez de depender de mocks ou simulações. Isso garante que os testes reflitam com mais precisão o comportamento da aplicação em produção.

Nota

Atenção à mapper Docker utilizada durante a execução dos testes de integração.

Dentre as demais dependências podem ser destacadas as seguintes, além daquelas já definidas no Banco de Dependências:

- Spring Framework
- Spring Boot
- Spring Security
- Spring Data
- Spring Batch
- Spring Integration
- Spring Web Services
- Spring Cloud
- Spring Data Rest

## Testes Unitários

Utiliza-se majoritariamente bibliotecas para unitários, mayormente direcionadas com o uso do Spring Boot:

- JUnit
- Mockito
- JUnit Jupiter

## Bibliotecas de Uso Restrito

Bibliotecas, ou frameworks de desenvolvimento, que não podem ser utilizadas no desenvolvimento do PJe:

- Lombok
- AOP

Nota

Lombok é uma biblioteca bastante conhecida que não é recomendada. Para projetos a adotarem.

# Diretrizes de Desenvolvimento

A seguir, algumas diretrizes de desenvolvimento para aplicações Java:

### 1. Atenda aos Padrões Arquiteturais e de Projeto

- Busque ao máximo atender aos padrões arquiteturais definidos pelo time de arquitetura e de projeto estabelecidos nesta documentação.

Nota

Caso haja dúvidas sobre os padrões, entre em contato com a equipe de arquitetura.

### 2. Convenções de Nomenclatura

- Adote convenções de nomenclatura consistentes para classes, métodos, variáveis e pacotes.
- Use nomes descritivos e significativos que transmitam claramente a finalidade e a funcionalidade do seu elemento.

Nota

Para mais detalhes, veja a seção de padrões de nomenclatura.

### 3. Documentação de Código

- Escreva código com comentários claros e concisos.
- Documente métodos e classes públicas usando Javadoc.
- Inclua descrições de parâmetros, valores de retorno e exceções lançadas.

### 4. Tratamento de Exceções

- Implemente um tratamento de exceções adequado e consistente.
- Evite capturar exceções genéricas (Exception) sempre que possível.
- Lance exceções específicas que forneçam informações úteis sobre o erro.

### 5. Logging

- Utilize o Log4j para registrar informações importantes sobre o comportamento da aplicação.
- Defina níveis de log apropriados (DEBUG, INFO, WARN, ERROR) para diferentes tipos de mensagens.
- Evite logar informações sensíveis, como senhas ou dados pessoais.

### 6. Testes Unitários

- Escreva testes unitários para cada componente ou classe relevante.
- Use frameworks de teste como JUnit e Mockito.
- Garanta que os testes sejam independentes e executáveis em qualquer ordem.

### 7. Testes de Integração

- Implemente testes de integração para verificar a interação entre diferentes componentes.
- Use frameworks como Arquillian para facilitar a execução de testes em um ambiente real.

### 8. Revisão de Código

- Realize revisões de código regularmente para garantir a qualidade e a consistência do código.
- Use ferramentas de análise estática de código para identificar possíveis problemas.

### 9. Configuração Externa

- Evite o uso de valores de configuração estáticos em seu código. Em vez disso, use arquivos de configuração externos (como .properties ou .yaml) ou variáveis de ambiente para definir valores que podem mudar entre diferentes ambientes.
- Isso torna sua aplicação mais flexível e fácil de configurar em diferentes ambientes (desenvolvimento, teste, produção).

# Padrões de Desenvolvimento

Como o PJe é um projeto de desenvolvimento de alta MASSA, os principais padrões empregados são os de Projeto, Arquitetura e Integração definidos pelo GoF e pelo EIP. Além disso o projeto também é o REST (Representational State Transfer), o Domain Driven Design (DDD) além de padrões específicos para o desenvolvimento de componentes JSF.

Dentre eles, destacamos alguns padrões mais usados e desenvolvimentos de aplicações específicas para o PJe. Para mais detalhes, veja a seção de Padrões Arquiteturais e a seção de Padrões de Implementação.

## Estruturas de Pacotes

Uma parte com MVP (padrão Arquitetural) e DDD (padrão de projeto):

- br.jus.cnj.pje.[SISTEMA]: Pacote raiz da aplicação.
  - [SISTEMA].action: Classes MVP que fazem a ligação com a View
  - [SISTEMA].dao: Configuração de acesso aos dados
  - [SISTEMA].entidades: Entidades do MVC, que estarão no mapeamento relacional do ORM
  - [SISTEMA].manager: Classes de negócio do MVC
  - [SISTEMA].service: Serviços, também de negócio, mas que não se encaixam no manager
  - [SISTEMA].util: Utilitários. Métodos de uso frequente ou comuns no negócio
  - [SISTEMA].ws.rest: Serviços REST, que implementam a lógica de negócio
  - [SISTEMA].ws.soap: Serviços SOAP, que encapsulam as operações de negócio
  - [SISTEMA].ws.util: Utilitários para os serviços

Recomenda-se que todo código que for possível ser reutilizado em outros sistemas do PJe, deve ser colocado organizado no Nó de Componentes. Por exemplo:

- br.jus.cnj.pje.[SISTEMA].[ENTIDADE]
  - [ENTIDADE].action: Classes MVP que fazem a ligação com a View
  - [ENTIDADE].dao: Configuração de acesso aos dados
  - [ENTIDADE].entidades: Entidades do MVC, que estarão no mapeamento relacional do ORM
  - [ENTIDADE].manager: Classes de negócio do MVC
  - [ENTIDADE].service: Serviços, também de negócio, mas que não se encaixam no manager
  - [ENTIDADE].util: Utilitários. Métodos de uso frequente ou comuns no negócio

## Nomenclatura de Classes

A nomenclatura de classes deve seguir o padrão CamelCase, com a primeira letra de cada palavra maiúscula. Recomenda-se que as classes, além de estarem em um pacote específico e

- Controladores MVP: [ENTIDADE]Action
- Controle de Sessão: [ENTIDADE]Manager
- Modelos de Dados: [ENTIDADE]
- Serviços de Dados: [ENTIDADE]DAO ou [ENTIDADE]Service
- Serviços de Negócio: [ENTIDADE]Manager
- Configurações: [ENTIDADE]Config

Entre as nomenclaturas de controladores devem ser incluídas aos controles correspondentes as anotações @Name("[ENTIDADE]Action") e @Scope(ScopeType.CONVERSATION) ou @Scope(ScopeType.EVENT) de acordo com as convenções vistas. Exemplo:

```
@Name("[ENTIDADE]Action")
@Scope(ScopeType.CONVERSATION)
public class [ENTIDADE]Action extends BaseAction<[ENTIDADE]> {
    ...
}
```

## Mapeamento JPA

Como regra geral:

- A classe/Pacote Base Validation deve ser criada para validar atributos em operações de persistência.
- Os métodos devem serem apenas para efetuarem atualizações conforme o mandato do negócio.
- Entidades devem derivarem uma única entidade de base (ex: [ENTIDADE]) ou, caso não haja, uma classe abstrata que contenha os atributos comuns a todas as entidades.
- Entidades devem ser anotadas com @Entity e @Table(name="[ENTIDADE]") e devem ser mapeadas com @Id e @GeneratedValue(strategy=GenerationType.IDENTITY) para atributos com sequences de postgresql.
- Entidades devem ser anotadas com @Entity e @Table(name="[ENTIDADE]") e devem ser mapeadas com @Id e @GeneratedValue(strategy=GenerationType.IDENTITY) para atributos com sequences de postgresql.

Exemplo de Implementação

Por exemplo, a classe [ENTIDADE] deve ser anotada com @Entity e @Table(name="[ENTIDADE]") e deve ser mapeada com @Id e @GeneratedValue(strategy=GenerationType.IDENTITY) para atributos com sequences de postgresql.

```
@Entity
@Table(name="[ENTIDADE]")
@GeneratedValue(strategy=GenerationType.IDENTITY)
public class [ENTIDADE] {
    ...
}
```

Nesse

# Padrões de Desenvolvimento Web

O Base abstrai o Vue.js como framework padrão por sua eficiência e qualidade no desenvolvimento de sistemas web, graças à sua simplicidade e flexibilidade. O uso do Quasar Framework, baseado no Vue.js, otimiza ainda mais o processo, pois oferece uma série de componentes prontos e um sistema de layout responsivo, facilita a criação de interfaces gráficas.

Além disso, a integração do TypeScript, uma linguagem de programação que adiciona tipagem estática ao JavaScript, melhora a robustez do código e facilita a manutenção. O TypeScript oferece recursos avançados como tipagem estática, interfaces, classes e genéricos, que aprimoram a qualidade e a manutenibilidade do código.

Para centralizar a Amazônia Framework disponibiliza um módulo chamado 'base-vue', que fornece componentes e serviços comuns para o desenvolvimento de sistemas web.

## Versões Suportadas

Priorizamos o uso de versões LTS (Long Term Support) dos frameworks e bibliotecas, para garantir a estabilidade e segurança a longo prazo. A tabela a seguir apresenta as versões suportadas pelo Base:

| Framework/Biblioteca | Versão Suportada |
|---|---|
| Vue.js | 3.4.x |
| Quasar Framework | 2.0.x |
| Base Vue | 2.1.8 |
| TypeScript | 5.x |
| Node.js | 20.x |

O Nginx é o servidor padrão para hospedar aplicações web no Base. Utilize imagens oficiais do Nginx disponibilizadas pela Red Hat no Red Hat Container Catalog.

## IDEs Recomendadas

O Visual Studio Code é a IDE recomendada para o desenvolvimento de sistemas web no Base. Além de ser uma ferramenta gratuita e de código aberto, o Visual Studio Code oferece uma série de extensões que facilitam o desenvolvimento, como o suporte ao TypeScript e ao Vue.js.

Recomende-se a instalação das seguintes extensões:

- Vue - Official
- Quasar Snippets
- ESLint
- Prettier - Code formatter
- SCSS Formatter
- Git History | GitLens | Git Graph

## Ferramentas de Build

No contexto de Ferramentas de Build para projetos Vue.js no Base, recomendamos o uso do Vite como package manager e do Vite como framework de build.

Segue abaixo como configurar a instalação interna do Nexus Repository para uso do YARN:

*Atente para o uso do caractere '.' ao final da URL*

```bash
npm i -g yarn # instalação do YARN
npm config set registry https://nexus.adm.bancoamazonia.sa:8443/repository/npm-group/
yarn config set registry https://nexus.adm.bancoamazonia.sa:8443/repository/npm-group/

npm config set "strict-ssl" false -g
yarn config set "strict-ssl" false -g

# Opcional:
yarn config set version-git-tag false # Desativa criação de novos commits usando 'yarn publish'
yarn config set version-tag-prefix "" # Substitui o prefixo padrão para versões no arquivo package.json
```

## Principais Dependências Externas

No Banco da Amazônia, o uso do Vue.js, Quasar Framework, e do módulo base-vue é obrigatório para o desenvolvimento de sistemas web.

Para adicionar o módulo base-vue ao projeto, execute o seguinte comando:

```bash
yarn add base-vue
```

Além disso, recomendamos o uso de outras bibliotecas e ferramentas para otimizar o desenvolvimento e garantir a qualidade do código. Abaixo seguem as principais dependências externas recomendadas:

- Axios para acesso a APIs
- date-fns para formatação de datas. Evite o uso do Moment.js.

## Criando um Novo Projeto

Para o bootstrap de um novo projeto, deve-se utilizar o Quasar CLI conforme documentação da ferramenta, observando as seguintes configurações:

- Build Tool: Vite
- Lint: prettier
- Testes Unitários: Jest
- CSS Pre processor: SASS (com formatação SCSS)
- Icons: Material Icons e, opcionalmente Font Awesome
- Pacote de Línguas: pt-BR

## Testes End-to-End (E2E) com Cypress

O Cypress é uma ferramenta de teste de ponta a ponta (end-to-end) que permite escrever e executar testes automatizados para aplicações web. Ele oferece uma API intuitiva e fácil de usar para interagir com elementos da interface do usuário, simular ações do usuário e verificar o comportamento da aplicação em diferentes cenários. Considere o BDD (Behavior Driven Development) ao escrever testes E2E com Cypress.

**Informações Extras**

Para maiores detalhes sobre a construção dos testes, consulte a documentação oficial do Cypress:

- Construindo testes e2e
- Construindo teste de componentes

Para os testes de componente, cada teste deve ter o mesmo nome, pasta e extensão (.cy.js) do componente que está sendo testado. Por exemplo, se você tem um componente chamado CreateCategoryButton.vue, o teste correspondente deve ser chamado CreateCategoryButton.cy.js.

Para utilizar o Cypress, é necessário instalar a ferramenta e configurar os testes no projeto. A instalação do Cypress pode ser feita com o comando:

```bash
yarn add cypress -dev
```

Após a instalação, o Cypress pode ser executado com o comando yarn cypress open para abrir a interface gráfica do Cypress, ou yarn cypress run para executar os testes em modo headless (sem interface gráfica).

Ao razz do projeto, crie o arquivo cypress.config.js com o seguinte conteúdo:

```javascript
const { defineConfig } = require("cypress");

module.exports = defineConfig({
  projectId: "SIGLA-DO-PROJETO", // <-- substitua pela sigla do projeto
  e2e: {
    setupNodeEvents(on, config) {
      // implement node event listeners here
    },
  },
});
```

## Configuração do SonarQube

Para permitir que o processo de CI/CD realize a análise de código, o projeto deve ser configurado para enviar os relatórios de cobertura de testes e de análise estática para o SonarQube. Para isso, execute os comandos abaixo na raiz do projeto:

```bash
yarn add -D sonarqube-scanner
yarn add -D browserslist@latest
yarn browserslist --update-db
```

Em seguida, crie um arquivo chamado sonarqube-scanner.config.js na raiz do projeto com o seguinte conteúdo:

**Arquivo sonarqube-scanner.config.js**

**Dica**

Para prevenir a inclusão de arquivos indesejados no controle de versão, assegure-se de que os seguintes itens estão listados no arquivo .gitignore. Se não estiverem, adicione-os:

- sonar-report.json
- .scannerwork

## Configuração de CI/CD

Para incluir um projeto no fluxo de CI/CD, é necessário criar na raiz do projeto um arquivo nomeado .gitlab-ci.yml com o seguinte conteúdo:

```yaml
include:
  - project: '$PIPELINE_TEMPLATE_PROJECT_PATH'
    ref: master
    file:
      - '$PIPELINE_WORKFLOW'
      - '/templates/web-template.gitlab-ci.yml'

# Construindo build do projeto

build-app:
  image: build
  image: $PL_REGISTRY_URL/$PL_DOCKER_NAMESPACE/docker-node20-vue3-quasar:$ONE_TAG_NODE_20
  extends: .setup-build-app
  script:
    - yarn install
    - quasar build
  artifacts:
    paths:
      - dist/*
    expire_in: 2 hours

# Realizando deploy do projeto

deploy-app:
  stage: deploy
  image: $PL_REGISTRY_URL/$PL_DOCKER_NAMESPACE/openshift-client:$ONE_TAG_OPENSHIFT
  extends: .setup-common-deployment
  before_script:
    - login-ocp
  script:
    - oc start-build $PL_BUILD_NAME --from-dir=dist/spa
  dependencies:
    - build-app
```

O comando de build para aplicações legadas em Vue.js 2 e Quasar 1 é yarn run build --mode $PL_PROFILE.

O job build-app realiza a construção do artefato de deployment. As imagens disponíveis, por enquanto do Node.JS, são:

- NodeJS 16 : $PL_REGISTRY_URL/$PL_DOCKER_NAMESPACE/docker-node16-vue3-quasar:$ONE_TAG_NODE_16
- NodeJS 20 : $PL_REGISTRY_URL/$PL_DOCKER_NAMESPACE/docker-node20-vue3-quasar:$ONE_TAG_NODE_20

Esses jobs podem ser personalizados de acordo com as necessidades do projeto. Os outros jobs, definidos internamente na ferramenta, configuram o ambiente de deployment e geram indicadores para o processo de CI/CD como um todo.

## Testes E2E com Cypress

Quando necessário, é possível incluir testes automatizados no processo de CI/CD. Utilize o contexto parcial do arquivo .gitlab-ci.yml a seguir como exemplo para adicionar testes E2E com o Cypress Framework e ajuste conforme a necessidade do projeto:

**Job para adição de testes E2E com Cypress**

# Padrões de Desenvolvimento

## Organização das Pastas da Aplicação

O conjunto de pastas para cada projeto que utiliza este modulo deve seguir a seguinte estrutura:

**Estrutura de Pastas**

| Arquivos/Pastas Importantes | Descrição |
|---|---|
| /public | Arquivos padrões para o Vue |
| /src/assets | Arquivos padrões para sistemas |
| /src/classes | Local para criação de classes para reuso de objetos e/ou entidades |
| /src/components | Pasta para criação de componentes específicos das aplicações |
| /src/services | Pasta para integrações externas, como back-ends REST |
| /src/styles | Pasta para arquivos de estilo |
| /src/pages | Pasta para as páginas de visualização/interface com o usuário |
| /src/cypress/e2e | Pasta para testes unitários do cypress |

## Diretrizes de Desenvolvimento

Ao desenvolver aplicações web com TypeScript, é importante seguir algumas diretrizes para garantir a qualidade, segurança e manutenibilidade do código. Abaixo estão algumas práticas recomendadas para o desenvolvimento de aplicações web com TypeScript:

1. **Configuração do TypeScript:**
   - Utilize sempre a versão mais recente do TypeScript disponível.
   - Configure o arquivo tsconfig.json de forma a habilitar as opções de compilação que melhor se adequam ao seu projeto e suas necessidades específicas.
2. **Tipagem Forte:**
   - Aproveite ao máximo as vantagens do TypeScript fornecendo tipos para todas as variáveis, parâmetros de função e retornos de função.
   - Evite o uso do tipo any sempre que possível, pois isso pode anular os benefícios da verificação de tipo do TypeScript.
   - Use interfaces e tipos personalizados para descrever objetos complexos e estruturas de dados.
3. **Utilize as Features do ES6 e além:**
   - Aproveite as funcionalidades do ES6 e versões posteriores do ECMAScript, como arrow functions, const e let, em vez de var, desestruturação, entre outras.
   - Utilize módulos ES6 (import/export) em vez de sistemas de módulos mais antigos, como CommonJS.
4. **Utilize Ferramentas de Análise Estática de Código:**
   - Integre ferramentas de análise estática de código, como ESLint e TSLint, para identificar possíveis problemas e manter a consistência do código.
   - Configure as regras do prettier para garantir boas práticas de codificação e conformidade com o padrão de estilo adotado.
5. **Testes Automatizados:**
   - Escreva testes unitários e de integração para todas as funcionalidades importantes da sua aplicação.
   - Utilize a ferramenta de teste Jest para executar e automatizar testes.
   - Considere o uso de ferramentas de teste de ponta a ponta (end-to-end) para garantir a funcionalidade correta da aplicação em diferentes cenários. O Cypress é uma excelente opção para testes E2E.
6. **Gestão de Dependências:**
   - Utilize o Yarn como gerenciador de pacotes do projeto.
   - Mantenha as dependências atualizadas regularmente para garantir que você esteja aproveitando as últimas correções de bugs e melhorias de segurança.
7. **Segurança:**
   - Evite vulnerabilidades conhecidas, como injeção de código, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), etc.
   - Valide e sanitize todas as entradas do usuário. Considere a Política de Segurança para mais detalhes.
   - Verifique se seu projeto está corretamente configurado para utilização do SonarQube para análise de vulnerabilidades.
8. **Documentação:**
   - Mantenha a documentação do código atualizada e detalhada, explicando o propósito de cada função, classe e módulo.
   - Opcionalmente, utilize comentários JSDoc para documentar os tipos de parâmetros, tipos de retorno e qualquer outra informação relevante.
9. **Padrões de Codificação:**
   - Adote padrões de codificação consistentes em todo o projeto. Isso inclui convenções de nomenclatura, estilo de código, indentação, etc.
   - Considere a adoção de um guia de estilo de código, como o Airbnb JavaScript Style Guide, e adapte-o conforme necessário para atender às necessidades do seu projeto.
10. **Revisão de Código:**
    - Realize revisões de código regularmente para garantir que o código esteja em conformidade com os padrões estabelecidos, seja seguro e eficiente.
    - Encoraje a colaboração e o feedback entre os membros da equipe durante o processo de revisão de código.

## Arquivo ".gitignore"

O arquivo .gitignore é um arquivo de configuração que especifica arquivos e diretórios que o Git deve ignorar. Ele é útil para evitar que arquivos temporários, arquivos de compilação e outros arquivos desnecessários sejam incluídos no seu repositório Git.

A seguir, o conteúdo do arquivo .gitignore para projetos Vue.js:

**Arquivo .gitignore Padrão**



Made with
Material for MkDocs

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



Arquitetura Corporativa
Introdução
Padrões Arquiteturais
Políticas de Segurança
Configuração e Governança
Fluxo CI/CD
APIs REST

Padrões de Desenvolvimento
Java
Web
Nomenclatura para Banco de Dados
Configuração OpenShift

# Nomenclatura de Objetos de Banco de Dados

Este documento estabelece regras de atribuição de nomes para objetos de banco de dados.

## Nomeação Geral de Objetos

Os padrões gerais descritos para objetos são aplicáveis a todos os objetos de banco de dados, salvo adição de características específicas para cada tipo de objeto.

Respeitando características do Oracle, nomes de objetos não deve exceder ***30 (trinta) caracteres***, não podendo iniciar com caractere numérico e não deve conter caracteres especiais, exceto o "_" (underscore) que deve utilizado para substituir o caractere em branco.

**Exemplo: Espaço em branco**

FORNECEDOR NACIONAL → FORNECEDOR_NACIONAL

Sempre que possível, deve-se utilizar qualificadores para refinar o significado do componente, evitando possíveis ambiguidades

**Exemplo: Qualificador**

AGENCIA + (qualificador: do banco) → AGENCIA_BANCARIA

A nomeação de tabelas associativas, oriundas de relacionamentos muitos-para-muitos, deve ser feita concatenando o nome das duas tabelas envolvidas no relacionamento em questão ou utilizando um nome que represente a associação entre as tabelas.

**Exemplo: Tabela Associativa**

USUARIO + SISTEMA → USUARIO_SISTEMA
USUARIO + CURSO → MATRICULA_ALUNO

Sempre que possível, deve-se evitar a redução dos nomes das tabelas e views, a fim de manter a clareza e a compreensão do significado do objeto. Caso a redução seja necessária, deve-se priorizar as formas curtas naturais (abreviações ou siglas) ou abreviações universais, evitando a criação de siglas ou abreviações próprias.

Nomes com até 5 (cinco) caracteres devem ter como abreviatura o próprio termo.

**Exemplo: Redução de Nomes**

CPF → CADASTRO DE PESSOA FÍSICA
IOF → IMPOSTO SOBRE OPERAÇÕES FINANCEIRAS
PJ → PESSOA JURÍDICA
dia → DIA

Quanto não houver formas curtas naturais, devem-se utilizar a seguinte regra para as formas curtas "derivadas", desde que a abreviatura resultante não corresponda à outra palavra na língua portuguesa:

- A abreviatura será composta pela primeira letra do termo, acrescida de duas consoantes consecutivas, retiradas do termo da esquerda para a direita, acrescida das últimas duas letras do termo, independente de serem consoantes ou vogais.
- Caso o termo não possua duas consoantes (retirando a primeira e as duas últimas letras), então deverá ser concatenado com o número de consoantes existentes (ou não) mais as vogais anteriores as duas últimas letras.

**Exemplo: Redução de Nomes**

SALÁRIO → SLRIO
PAGAMENTO → PGMTO
ALTERAÇÃO → ALTAO
RECEITA → RCITA - (poucas consoantes)

## Colunas

Nomes para colunas devem seguir as mesmas regras de nomeação de objetos definidos anteriormente, com o acréscimo de prefixo que identifique a característica de dado que a coluna armazena.

Abaixo segue a lista de prefixos que devem ser utilizados para identificar a característica do dado armazenado na coluna:

| Prefixo | Descrição | Exemplo |
|---|---|---|
| ID | Identificador único, reservado para chaves primárias | ID_USUARIO |
| TX | Texto | TX_NOME |
| DT | Data, ou parte de data | DT_NASCIMENTO, DT_ANO_REGISTRO, DT_BASE_PAGAMENTO |
| DH | Data e Hora, fuso horário opcional | DH_CRIACAO |
| HR | Hora | HR_INICIO_JORNADA |
| IN | Indicador, geralmente booleano | IN_ATIVO |
| EN | Enumerador, geralmente de domínio restrito | EN_ESTADO_CIVIL |
| NU | Numérico, geralmente inteiro, sequencial ou quantitativo | NU_IDADE |
| VL | Valor, geralmente fracionário ou monetário | VL_SALARIO |
| BI | Binário | BI_FOTO |

## Constraints

Nomes para colunas devem seguir as mesmas regras de nomeação de objetos definidos anteriormente, com o acréscimo de prefixo que seguem os seguintes critérios:

- Caso exista mais de uma constraint de mesmo tipo na tabela, deve-se colocar um número seqüencial de duas posições após o prefixo;
- A nomeação das restrições de atribuição de único valor (unique) e de validação (check) devem ser compostas pelo prefixo, nome do objeto (tabela) e atributo ao qual a restrição é aplicada.

A tabela a seguir apresenta a nomenclatura de constraints:

| Prefixo | Descrição | Exemplo |
|---|---|---|
| PK | Primary key | PK_CD_CLIENTE |
| FK | Foreign key | FK01_CD_CLIENTE_PAGAMENTO |
| UK | Unique key | UK_EMPRESA_NOME |
| CK | Check constraint | CK_CLIENTE_SEXO |

## Índices

A nomenclatura dos índices deve ser composta pelo prefixo IX, número seqüencial, caractere especial underscore ("_") e o nome da tabela envolvida.

**Exemplo**

- IX01_CLIENTE
- IX02_PAGAMENTO

## Sequences

A nomeação de sequences deve ser composta pelo prefixo SQ, caractere especial underscore ("_") e mais o nome da tabela

**Exemplo**

- CLIENTE (tabela) → SQ_CLIENTE
- INDICADOR_FINANCEIRO (tabela) → SQ_INDICADOR_FINANCEIRO

## Sinônimos

Os sinônimos devem ser definidos pela área de Administração de Dados, sem a necessidade de codificação, sendo que os mesmos devem seguir o nome da tabela de origem do banco de dados.

## Functions, Procedures e Packages

Utilizar na nomenclatura os seguintes qualificadores de objeto:

| Prefixo | Descrição | Exemplo |
|---|---|---|
| FC | Function | FC_CALCULA_SALARIO |
| PR | Procedure | PR_ATUALIZA_ENDERECO |
| PC | Package | PC_CALCULA_IMPOSTO |

## Triggers

A nomenclatura destes objetos deve ser composta por um prefixo formado por sequencial de caracteres que identifique o comportamento da trigger, seguido do caractere especial underscore ("_") e o nome da tabela envolvida.

Este prefixo é composto por:

1. **T**: Indicando que o objeto é uma trigger;
2. **R** ou **S**: Nível de atuação da trigger (R - row ou S – statement);
3. **A** ou **B**: Tipo de acionamento da trigger (A – After ou B – Before);
4. **I**, **U** ou **D**: Tipo de operação efetuada (I – insert, U – update ou D – delete).

**Exemplo**

- TRAI_CLIENTE → Trigger AFTER ROW INSERT na tabela CLIENTE
- TRAIU_PAGAMENTO → Trigger AFTER ROW INSERT ou ROW UPDATE na tabela PAGAMENTO

## Código PL/SQL

Seguem as seguintes regras para nomenclatura de variáveis, cursores e demais objetos para código PL/SQL:

1. Para variáveis globais, dentro de um pacote, o nome da mesma deve iniciar com o prefixo `G_`;
2. Para variáveis locais, o nome deve iniciar com o prefixo `L_`;
3. Para cursores, o nome deve iniciar com o prefixo `C_`;
4. Para tipos (types) definidos pelo usuário, o nome deve iniciar com `T_`.

On this page
Nomeação Geral de Objetos
Colunas
Constraints
Índices
Sequences
Sinônimos
Functions, Procedures e Pac...
Triggers
Código PL/SQL



Made with Material for MkDocs
Created by AMAZONIA SOFTWARE

Manual de Desenvolvimento
Arquitetura Cooperativa
Introdução
Padrões Arquiteturais
Políticas de Segurança
Governança e Governança Corporativa
Conformidade
APIs REST
Padrões de Desenvolvimento
Java
Python
Nomenclatura para Banco de Dados
Configuração OpenShift

# OpenShift

O OpenShift é uma plataforma de contêineres baseada em Kubernetes, desenvolvida pela Red Hat, que oferece recursos adicionais para facilitar o desenvolvimento, a implantação e o gerenciamento de aplicativos em contêineres. Ele fornece uma camada de abstração e automação.

Este documento detalha as configurações e conexões de nomenclatura para objetos no OpenShift, incluindo como criar e acessar dois tipos de acesso a serviços: Rotas e Serviços. A forma orientadora sobre como interagir com o OpenShift utilizando a linha de comando.

> **Aviso**
>
> Apesar deste documento demonstrar exemplos de objetos no OpenShift, estes são criados automaticamente pelo pipeline GitOps, que é responsável por gerenciar a implantação de aplicações no OpenShift.

On this page
Clusters OpenShift Disponíveis
Instalação do OpenShift CLI
Objetos Kubernetes
Criação dos Projetos
Implantação de Novos Builds
Expor Serviço via Protocolo HTTP
Configurações a Secrets
Leitura de Logs
Nomenclatura de Objetos Da...

## Clusters OpenShift Disponíveis

No Beta, os ambientes OpenShift disponíveis atualmente são:

- OpenShift 1: https://cnsnp.apps.openshiftbeta.ca:6443
- OpenShift 4:
  - Console: https://console-openshift-console.apps.ocp.tanscomsistemas.ca/
  - Protocolo: https://console-openshift-console.apps.ocp.tanscomsistemas.ca/

## Instalação do OpenShift CLI

O OpenShift CLI (`OC`) é uma ferramenta de linha de comando que permite interagir com clusters OpenShift. Para instalação, siga os passos abaixo:

1. **Faça o download do OpenShift CLI**
   Você pode baixar o cli-dist oficial do OpenShift no site oficial do repositório específico. Para baixar do site oficial:
   - Visite a página de baixa do OpenShift no GitHub.
   - Faça o download do arquivo mais recente adequado para o seu sistema operacional. Para Linux, geralmente é o arquivo `oc.tar.gz`.

2. **Extrair o arquivo**
   Depois de baixar o arquivo, você precisará extraí-lo seu conteúdo. No Linux, você pode usar o seguinte comando:

```
tar -xvf oc.tar.gz
```

   Isso criará um diretório com o nome do cenário do OC que você baixou.

3. **Adicionar o `oc` ao PATH**
   Para usar o cliente OC globalmente no Linux, adicione o diretório do OC ao seu PATH do sistema, incluindo este linha no arquivo `~/.bashrc` ou `~/.zshrc` (dependendo do seu shell) e reinicie o seu sistema operacional. Por exemplo, se o cliente OC estiver em `~/.cache/oc/oc`:

```
export PATH=$PATH:~/.cache/oc/oc
```

   No Windows, você pode adicionar o diretório do OC ao PATH usando as Configurações Avançadas do Sistema. Para fazer isso, clique com o botão direito em "Este Computador", vá para "Configurações Avançadas do Sistema", e em seguida em "Variáveis de Ambiente". Adicione o diretório do OC ao PATH na seção "Variáveis de Sistema".

4. **Verificar a instalação**
   Após instalar o `oc` e adicioná-lo ao seu PATH, reinicie/abra seu arquivo de configuração de shell e abra um novo terminal. Agora você deve ser capaz de executar o comando `oc` para interagir com seu cluster OpenShift.

5. **Autenticação**
   Antes de iniciar o uso do cliente OC, é necessário autenticar-se no seu cluster OpenShift. Isso pode ser realizado fornecendo um token de acesso ou credenciais de usuário e senha. A autenticação é efetuada através do comando `oc login`. Veja um exemplo abaixo:

> **Dica**
>
> O username e password são os mesmos para acesso a rede interna de Banco da Amazônia, enquanto o token de acesso é obtido na web console do OpenShift no canto superior direito do cluster.

Com esses passos, você deve ser capaz de instalar e configurar o OpenShift CLI e usá-lo to it para interagir com os cluster.

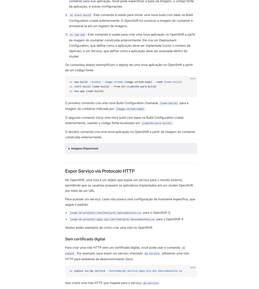
> **Once you Loss/Wiki 2**

## Objetos Kubernetes

O OpenShift é baseado no Kubernetes, e muitos dos conceitos e objetos do OpenShift são os mesmos que os do Kubernetes. Alguns dos objetos mais comuns do OpenShift incluem:

1. **Namespaces**: Um namespace é uma maneira de dividir recursos do OpenShift em grupos lógicos. Isso é útil para organizar e isolar recursos para diferentes projetos ou equipes, ou para controlar quem pode acessar o que.
2. **Build Configuration**: uma build configuration define como criar uma imagem de uma aplicação a partir de um código-fonte. Ela pode ser configurada para acionar builds automaticamente quando o código-fonte é alterado.
3. **Deployment Configuration**: Uma deployment configuration controla como um aplicativo é implantado e atualizado no OpenShift. Ela pode ser configurada para acionar implantações contínuas e também de rollbacks a estratégia de atualização.
4. **Service**: Um serviço é uma abstração que define um conjunto lógico de pods e uma política de acesso a eles. Ele permite que aplicativos se comuniquem uns com os outros no OpenShift sem conhecer os detalhes de implantação dos pods.
5. **Route**: Uma rota expõe serviços para o mundo externo, permitindo que os usuários externos acessem seus aplicativos. Ela fornece um nome de host e um caminho para acessar o serviço.
6. **Pod**: Um pod é a menor unidade de implantação no OpenShift. Ele representa um ou mais contêineres que são sempre implantados juntos em um só nó do OpenShift.
7. **Config/Secret**: Objeto que armazena dados de configuração, como variáveis de ambiente, arquivos de configuração e segredos, como senhas e chaves de API. Eles podem ser montados em contêineres de configuração no início do pod ou durante a execução.
   > O Config/Map é usado para armazenar configurações não confidenciais, enquanto o Secret é usado para armazenar informações sensíveis, como senhas e chaves de API.
8. **Volume**: É uma abstração representando uma área de armazenamento para um contêiner em um pod. Ele permite a acesso e compartilhamento de dados persistentes, como arquivos, entre contêineres. No OpenShift, volumes são usados para manter dados entre reinicializações de contêiner e fornecer armazenamento persistente para aplicativos.

O gráfico abaixo ilustra a relação entre estes objetos:




## Comandos Básicos de Leitura

Para obter informações sobre os objetos demonstrados, você pode usar o comando `oc get` seguido pelo tipo de objeto que deseja visualizar.

Por exemplo, para listar todos os serviços em um namespace, você pode usar:

```
oc get svc
```

Para obter mais informações sobre a versão, você pode utilizar uma das três formas demonstradas abaixo:

```
oc get svc -[nome do versão]
oc describe svc [nome do versão]
oc get [nome do versão] -o yaml
```

A primeira forma apresenta os atributos gerais do serviço, a segunda apresenta uma descrição mais detalhada e a terceira exibe os detalhes do serviço em formato YAML.

Abaixo estão os comandos para obter informações sobre os demais objetos:

- **Build Configuration**: `oc get bc`
- **Deployment Configuration**: `oc get dc`
- **Route**: `oc get route`
- **Pod**: `oc get pod`
- **Config/Map**: `oc get configmap`
- **Secret**: `oc get secret`
- **Service**: `oc get svc`

## Criação dos Projetos

No OpenShift, um projeto é um namespace do Kubernetes com recursos adicionais e restrições que permitem a um grupo de usuários gerenciar seus aplicativos isoladamente.

No Beta, cada aplicação é alocada em três namespaces de Banco da Amazônia, correspondentes aos ambientes de desenvolvimento, homologação e produção. Esses namespaces seguem o padrão `[nome]-dev`, `[nome]-hml` e `[nome]-prd`, respectivamente.

Os comandos abaixo exemplificam a criação de um novo projeto no OpenShift e a criação dos namespaces de homologação (hml):

```
oc new-project [nome-projeto] hml
oc label namespace [nome-projeto]-hml openshift.io/cluster-monitoring=true
```

O comando `oc project` é empregado para alternar um cliente ao projeto. Substituindo a seleção dos nós do cluster onde os pods serão implantados. Por exemplo, o rótulo `zone=dmz` direciona as instâncias dos microsserviços ao ambiente de homologação.

> **Aviso**
>
> O processo de criação de projetos no OpenShift é feito via Ansible, que é uma ferramenta de automação de TI de código aberto que permite a automação de configuração de sistemas, implantação de software e orquestração de tarefas de TI.

## Implantação de Novas Builds

As instâncias formas de implantar novas builds no OpenShift, mas a mais comum utilizada é a instrução a partir de um código-fonte. Para isso, é necessário criar uma nova Build Configuration (BC) que especifique o repositório de código-fonte, a estratégia de build e quais parâmetros devem ser utilizados.

Os comandos `oc new-build` , `oc start-build` e `oc new-app` são usados no OpenShift para criar e implantar novos aplicativos.

1. `oc new-build`: Este comando é usado para criar uma nova build com base no fluxo de trabalho especificado anteriormente. O OpenShift cria um novo build com base nesse fluxo e armazena-o em um registro de imagens.
2. `oc start-build`: Este comando é usado para criar uma nova build com base no fluxo de trabalho especificado anteriormente. O OpenShift cria um novo build com base nesse fluxo e armazena-o em um registro de imagens.
3. `oc new-app`: Este comando é usado para criar uma nova aplicação no OpenShift a partir de diversas fontes, incluindo código-fonte, imagens de contêiner e modelos. Ele cria uma nova configuração de build e implantação para o aplicativo e, opcionalmente, cria um serviço e uma rota para acessá-lo.

Os comentários abaixo exemplificam a criação de uma nova aplicação no OpenShift a partir de um código-fonte:

```
oc new-build --binary --image-stream [image-stream:tag] --name [name-build]
oc start-build [name-build] --from-dir=[caminho para build]
oc new-app [name-build]
```

O primeiro comando cria uma nova build com base na Build Configuration criada anteriormente, usando o código-fonte localizado em `[caminho para build]`.

O terceiro comando cria uma nova aplicação no OpenShift a partir da imagem de contêiner construída anteriormente.

> **Imagens Disponíveis**

## Expor Serviço via Protocolo HTTP

No OpenShift, um serviço é um objeto que expõe um serviço para o mundo externo. Ele fornece um endereço IP e porta estáveis para acessar o serviço, mesmo que os pods subjacentes sejam criados ou destruídos.

Para acessar um serviço, cada rota possui uma configuração de hostname específica, que segue o padrão:

- `[nome-do-projeto].[DOMÍNIO].[ambiente].tanscomsistemas.ca` para o OpenShift 1.
- `[nome-do-projeto].apps.ocp.[DOMÍNIO].[ambiente].tanscomsistemas.ca` para o OpenShift 4.

Abaixo estão exemplos de como criar uma rota no OpenShift:

### Sem certificado digital

Para criar uma rota HTTP sem certificado digital, você pode usar o comando `oc expose`. Por exemplo, para expor um serviço chamado `ag-service` utilizando uma rota HTTP para ambientes de desenvolvimento (dev):

```
oc expose svc/ag-service --hostname=ag-service.apps.ocp.tcs.tanscomsistemas.ca
```

Isso criará uma rota HTTP que mapeia para o serviço `ag-service`.

> **Aviso**
>
> Para aplicações do HTTP sem TLS (não é desejável. Utilize apenas para testes locais).

### Com certificado digital

Para criar uma rota HTTP com um certificado digital (HTTPS), você precisará criar um arquivo de configuração de rota que especifique o certificado e a chave privada. Você pode usar o seguinte exemplo de configuração YAML. Após isso crie exemplo:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: ag-service
spec:
  host: ag-service.apps.ocp.tcs.tanscomsistemas.ca
  port:
    targetPort: 8080-tcp
  tls:
    key: |-
      -----BEGIN PRIVATE KEY-----
      [KEY]
      -----END PRIVATE KEY-----
    termination: edge
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [CERTIFICATE]
      -----END CERTIFICATE-----
    insecureEdgeTerminationPolicy: Redirect
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [CA CERTIFICATE]
      -----END CERTIFICATE-----
  certificates:
    - |-
      -----BEGIN CERTIFICATE-----
      [CA CERTIFICATE]
      -----END CERTIFICATE-----
  to:
    kind: Service
    name: ag-service
```

Substitua `[KEY]`, `[CERTIFICATE]` e `[CA CERTIFICATE]` pelos seus valores reais.

Para criar uma rota HTTPS que mapeia para o serviço `ag-service` e usa o certificado fornecido.

### Configurações Extras para Rota

1. Para aumentar o timeout do comando HTTP de rota, por exemplo para dois minutos, utilize o seguinte comando:

```
oc annotate route [name-da-rota] --overwrite haproxy.router.openshift.io/timeout=2m
```

## Configuração a Secrets

Config/Maps e Secret são dois tipos de recursos no OpenShift que permitem gerenciar a configuração e os dados confidenciais de aplicativos. Eles são:

**Config/Maps**: Um recurso que permite armazenar configurações não confidenciais como pares de chave-valor. Ele pode ser usado para armazenar coisas como URLs de ponto de extremidade, números de porta, configurações de configuração e muito mais. Os aplicativos podem acessar os dados em um Config/Map como variáveis de ambiente ou montando-o como um volume.

**Secret**: Um recurso que permite armazenar informações confidenciais como senhas, tokens e chaves de API. Os dados em um Secret são codificados em base64 e podem ser acessados de maneira semelhante aos Config/Maps. No entanto, os Secrets também oferecem recursos adicionais de segurança, como criptografia em repouso.

Abaixo de criar e acessar os configurações `oc create` e `oc apply` e `oc describe` Secret.

```
oc create secret generic ag-secret --from-literal=username=admin --from-literal=password=password
```

Neste exemplo, um Secret chamado `ag-secret` é criado com duas entradas: `username` e `password`.

Por padrão, o aparelho CSTO gera um Config/Map e um Secret, nomeados como `[nome-do-projeto]-config` e `[nome-do-projeto]-secret`, respectivamente. Esses recursos são automaticamente usados para expor todos os seus atributos como variáveis de ambiente para aplicativos. Isso permite a definição de variáveis de configuração específicas para cada ambiente (dev, hml, prd) sem precisar alterar o código dependendo do ambiente de destino.

Como exemplo, segue abaixo comandos para configurar estes objetos como variáveis e variáveis de ambiente:

```
oc set volume deployment --add --overwrite volume --mount-path=/etc/config --configmap-name=ag-config
oc set env deployment/ag-app --from=configmap/ag-config
```

## Leitura de Logs

Para obter os logs dos pods no OpenShift, você pode usar o comando `oc logs`. Aqui está como você pode usá-lo para obter os logs de um pod:

1. `oc get pod`
   Isso listará todos os pods no seu projeto atual. Você verá uma saída semelhante a esta:

```
NAME                     READY   STATUS    RESTARTS   AGE
api-instance-77f-kyhv9   1/1     Running   0          10d3h
api-instance-77f-qzxwl   1/1     Running   0          10d3h
release-1-deploy         0/1     Completed 0          10d3h
```

2. `oc logs api-instance-77f-kyhv9`

No exemplo acima, o comando `oc get pod` é usado para listar os pods em um namespace. Em seguida, o comando `oc logs` é usado para obter os logs do pod `api-instance-77f-kyhv9`.

Você também pode obter logs de implantações e compilações usando o mesmo comando. No exemplo abaixo, os logs de uma compilação (Build Configuration) e de uma implantação (Deployment Configuration) são obtidos.

```
oc logs bc/api-instance
oc logs dc/api-instance
```

Duas opções muito úteis para facilitar a leitura e o acompanhamento dos logs são `-f` e `--tail`:

- `-f` (`follow`): A saída será "seguir" os logs em tempo real. Quando você executa `oc logs -f [nome-do-pod]`, o comando exibirá os logs existentes e, em seguida, aguardará novas entradas e as exibirá à medida que forem adicionadas ao pod. Isso é útil para monitorar o aplicativo ou depuração em tempo real. Por exemplo:

```
oc logs -f [nome-do-pod]
```

- `--tail`: O `--tail` permite que você especifique quantas linhas dos logs mais recentes deseja ver inicialmente. Isso é útil quando você só precisa das informações mais recentes e não quer carregar uma grande quantidade de logs antigos. Por exemplo, para exibir apenas as últimas 10 linhas de logs de um pod:

```
oc logs --tail=10 [nome-do-pod]
```

Ao combinar essas opções com o comando `oc logs`, você pode controlar melhor a quantidade de dados exibidos e acompanhar os logs de forma mais eficiente, especialmente ao lidar com cenários de desenvolvimento, teste e diagnóstico.

## Nomenclatura de Objetos OpenShift

O namespace é o objeto de mais alto nível superior no OpenShift, que é responsável por agrupar e isolar outros objetos. Ele é usado para separar diferentes projetos ou equipes no mesmo cluster. O nome do namespace deve seguir o padrão `[nome-do-projeto].[DOMÍNIO]`.

Dependendo do ambiente de destino, os objetos no OpenShift são nomeados de forma diferente. Para os branches padrões (`build`, `release` e `master`), o padrão de nomenclatura para cada objeto segue abaixo:

**Branches padrões**

- **Build Configuration**: nome do projeto
- **Deployment Configuration**: nome do projeto
- **Route**: nome do projeto
- **Hostname**: nome do projeto + `.apps.ocp.[DOMÍNIO].tanscomsistemas.ca`
- **Configmap**: nome do projeto + `-configmap`
- **Secret**: nome do projeto + `-secret`

Para branches especiais (exclusivas nomeadas como *feature* (`feature`) ou *release* (`hml`)), o padrão segue abaixo:

**Branches Especiais/outros**

- **Build Configuration**: nome da branch
- **Deployment Configuration**: nome da branch
- **Route**: nome da branch
- **Hostname**: nome do projeto + nome da branch + `.apps.ocp.[DOMÍNIO].tanscomsistemas.ca`
- **Configmap**: nome do projeto + `-configmap`
- **Secret**: nome do projeto + `-secret`

Por exemplo, para um projeto chamado `ag-service`, os objetos para a branch master seriam nomeados como:

- **Build Configuration**: ag-service
- **Deployment Configuration**: ag-service
- **Route**: ag-service
- **Hostname**: ag-service.apps.ocp.prd.tanscomsistemas.ca
- **Configmap**: ag-service-configmap
- **Secret**: ag-service-secret

Para a branch *feature* nova-funcionalidade, os objetos seriam nomeados como:

- **Build Configuration**: feature-nova-funcionalidade
- **Deployment Configuration**: feature-nova-funcionalidade
- **Route**: feature-nova-funcionalidade
- **Hostname**: ag-service.feature-nova-funcionalidade.apps.ocp.prd.tanscomsistemas.ca
- **Configmap**: ag-service-configmap
- **Secret**: ag-service-secret



Made with Material for MkDocs

# ANEXO II - DOCUMENTAÇÃO DO AMAZÔNIA FRAMEWORK

**Amazônia Framework**
Introdução
Termo de Uso

**Java - Versão 4**
Configurações Gerais
APIs REST
Apache Camel

**Web - Versão 2**
Configurações Gerais
Componentes Vue.js
Ferramentas de Desenvolvimento

**Java - Versão 3 (Legado)**

**Web - Versão 1 (Legado)**

# Amazônia Framework

On this page
Plataforma Java
Plataforma Web

O Amazônia Framework é uma coleção interna de bibliotecas e ferramentas do Banco da Amazônia, visando agilizar o desenvolvimento de sistemas, fornecer soluções testadas para requisitos não-funcionais e padronizar a arquitetura dos sistemas para facilitar a manutenção e o repasse de conhecimento entre os fornecedores e analistas.

Possui bibliotecas para *Java* e e plataforma Web com *VueJS*, com potencial para expansão conforme a arquitetura do Banco evolui e novas plataformas de desenvolvimento são adotadas.

## Plataforma Java

O *Amazônia Framework* para Java utiliza o ***Spring Boot*** como base de desenvolvimento, aplicando uma série de autoconfigurações nos projetos buscando integrá-los de maneira transparente à infraestrutura do Banco. Além disso, disponibiliza ferramentas a fim de prover o máximo de funcionalidades e reaproveitamento de código entre projetos.

**Dica**

Acesse o site oficial do Spring Framework para mais informações sobre seu ecossistema de ferramentas.

### Versões

A *major version* atual do *Amazônia Framework* para Java é a 4. A documentação da versão 3 ainda é mantida para projetos legados. Abaixo segue as versões para as principais bibliotecas utilizadas:

| Biblioteca | Versão |
|---|---|
| Amazônia Framework | 4.1.9 |
| Spring Boot | 3.3.1 |
| Spring Cloud | 2023.0.2 |
| Apache Camel | 4.4.2 |

## Plataforma Web

O *Amazônia Framework* para Web utiliza o ***Vue.js*** como base para desenvolvimento de interfaces web e single-page applications (SPA). O módulo `basa-vue` disponibiliza ferramentas comuns para formatação e validação de dados, módulos de Segurança integrado com o RH-SSO e AmazonSegu, padrões visuais e alguns componentes de input de dados.

### Versões

A *major version* atual do *Amazônia Framework* para Web é a 2. A documentação da versão 1 ainda é mantida para projetos legados. Abaixo segue as versões para as principais bibliotecas utilizadas:

| Biblioteca | Versão |
|---|---|
| Amazônia Framework | 2.1.8 |
| VueJS | 3.4.x |
| Quasar Framework | 2.0.x |
| TypeScript | 5.x |
| NodeJS | 20.x |

Next page
Introdução

Mon Nov 25 2024 17:20:58 GMT-0300 (Brasilia Standard Time)
Created by JUST ONE PAGE PDF

Amazônia Framework
Introdução
Termo de Uso

Java - Versão 4
Configurações Gerais
APIs REST
Apache Camel

Web - Versão 2
Configurações Gerais
Componentes Vue.js
Ferramentas de Desenvolvimento

Java - Versão 3 (Legado)

Web - Versão 1 (Legado)

# Termo de Uso

On this page
TERMO DE USO PARA ASSIN...
TERMO DE USO

## TERMO DE USO PARA ASSINATURA

Download: Termo-de-Uso-do-Amazônia-Framework.pdf

Disponibilizamos o Framework em formato PDF para impressão, assinatura e posterior entrega via carta ao Banco da Amazônia. Esta solicitação aplica-se somente às empresas fornecedoras de serviços ao Banco. O termo de uso pode ser baixado do sistema de normativos, NP029, na área de anexos.

## TERMO DE USO

Este Termo de Uso do Framework para Desenvolvimento de Sistemas do Banco da Amazônia ("Amazônia Framework") regulamenta sua utilização por desenvolvedores terceirizados, disponibilizado pelo BANCO DA AMAZÔNIA S.A., empresa com sede na Cidade de Belém, no Estado do Pará, sito à Avenida Presidente Vargas, nº 800, inscrito no CNPJ/MF sob o nº 04.902.979/0001-44, doravante denominado simplesmente "Banco". No ato de acesso ao Amazônia Framework, o usuário (doravante denominado "Desenvolvedor de Sistemas" ou "Equipe Desenvolvedora de Sistemas") se obriga a aceitar, plenamente e sem reservas, todos os termos e condições aqui presentes:

1. DEFINIÇÕES

1.1. Para os efeitos deste instrumento os vocábulos e expressões abaixo têm as seguintes definições: Amazônia Framework – Plataforma para desenvolvimento de sistemas do Banco da Amazônia, não incluindo seu código fonte e os conhecimentos utilizados na sua produção; Backup – Cópia de segurança que deve ser mantida pelo Banco, tanto dos arquivos de instalação dos softwares, quanto dos arquivos e bases de dados; Suporte – É o atendimento aos desenvolvedores, cuja finalidade é esclarecer dúvidas de funcionamento do Amazônia Framework; Licença de Uso – É a autorização para a utilização do Amazônia Framework por prazo determinado (enquanto durar o contrato) quando se tratar de desenvolvimento externo (fornecedores e/ou fábrica de software) e indeterminado quando se tratar de desenvolvimento interno (equipe de empregados do Banco); Atualização de Versão – É a disponibilização de novas versões do Amazônia Framework; Capacitação – É o processo de transferência de conhecimento para os desenvolvedores que utilizarão o Amazônia Framework, por intermédio de sua documentação, para que os mesmos possam desenvolver os sistemas corretamente e de acordo com os padrões e infraestrutura estabelecidos pelo Banco da Amazônia.

2. OBJETO

2.1. O objeto do presente contrato é a licença de uso do Amazônia Framework por prazo determinado (enquanto durar o contrato) quando se tratar de desenvolvimento externo (fornecedores e/ou fábrica de software) e indeterminado quando se tratar de desenvolvimento interno (equipe de empregados do Banco).

3. DISPOSIÇÕES GERAIS

3.1 O presente Contrato de Licença será fornecido impresso, para ciência e assinatura, quando da assinatura de contrato com empresa prestadora de serviços de desenvolvimento de sistemas e quando da contratação ou alocação de empregados para trabalharem com desenvolvimento de sistemas. Sendo assim, as equipes (internas ou de fornecedores) terão ciência de todos os recursos disponíveis no objeto do presente documento e de que este foi desenvolvido de forma a atendê-los de forma geral, não estando sujeita ao Banco da Amazônia providenciar alterações exclusivas para nenhum desenvolvedor e nem a posteriores reclamações. A simples utilização do Amazônia Framework vincula o desenvolvedor à aceitação e cumprimento de todos os itens aqui presentes.

3.1.1. A inviabilidade do cumprimento de algum item deste Contrato de Licença, ou sua invalidade, ainda que em virtude de decisão judicial, não afetará a validade dos demais itens, permanecendo este documento válido e executável conforme descrito em seus demais itens;

3.1.2. As atualizações do Amazônia Framework disponibilizadas pelo Banco da Amazônia para os desenvolvedores poderão conter termos adicionais, que, uma vez utilizadas, serão considerados como se de fato foram lidos e passarão a compor este Contrato de Licença de forma indissociável. Dessa forma, fica sob responsabilidade dos desenvolvedores ler os termos adicionais nas atualizações do Amazônia Framework.

3.2. Os defeitos porventura identificados no Amazônia Framework serão corrigidos, necessariamente, pelo Banco da Amazônia, sendo que as novas versões serão disponibilizadas aos desenvolvedores.

3.3. Os desenvolvedores que, porventura, identificarem algum problema no Amazônia Framework deverão comunicar ao Banco da Amazônia através de sua coordenadoria responsável pela Arquitetura dos Sistemas de TI, para providências das devidas correções.

3.4. O Banco da Amazônia poderá disponibilizar novas versões do Amazônia Framework com outros recursos, cuja utilização, onerosa, ficará a critério dos desenvolvedores.

4. CONDIÇÕES DE USO

4.1. O Amazônia Framework utiliza componentes, bibliotecas e módulos públicos e que possuem Licença Pública Geral GNU, ou seja, que podem ser utilizados aberta e livremente. Contudo, também é dotado de módulos próprios, desenvolvidos pelo Banco, especificamente para os propósitos intrínsecos aos negócios da instituição, os quais estão protegidos e respaldados por contrato de licença de uso e de propriedade intelectual, não sendo permitido seu uso por terceiros sem prévia autorização formal e, ainda que permitido, devem ser utilizados única e exclusivamente em sistemas desenvolvidos para o Banco da Amazônia.

4.2. Esta Licença permite a utilização do Amazônia Framework e seus componentes próprios, no que se refere única e exclusivamente ao desenvolvimento de sistemas para o Banco, não sendo permitido modificá-lo sob quaisquer hipóteses.

4.3. O direito de uso por este contrato é vigente pela duração do contrato estabelecido com fornecedor, ou equipe desenvolvedora, interna (empregados do Banco) ou externa (terceirizados), que se fará valer do Amazônia Framework para desenvolvimento de sistemas do Banco da Amazônia, e são improrrogáveis logo que as condições estabelecidas sejam cumpridas.

4.4. Todos os sistemas desenvolvidos no Amazônia Framework, seja por equipe do Banco ou por fornecedores, estão cobertos por esta Licença.

5. RESTRIÇÕES

5.1. O presente Contrato de Licença é regido pelas disposições da Lei 9.609/98, ficando os infratores sujeitos às penas dos crimes previstos no art. 12, sem prejuízo da responsabilidade civil pelos danos eventualmente causados pelo uso e distribuição de cópias não autorizadas do Amazônia Framework ou por qualquer outra violação aos direitos decorrentes da propriedade do Amazônia Framework.

5.1.1 O Amazônia Framework é protegido por leis e tratados internacionais de direitos autorais e de propriedade intelectual. A titularidade de todos e quaisquer direitos autorais e de propriedade intelectual sobre o Amazônia Framework é do Banco da Amazônia. É disponibilizada a licença de uso do Amazônia Framework aos desenvolvedores pelo prazo expresso no item 2.1, e não há qualquer cessão de direitos relativos à titularidade do mesmo.

5.1.2 É vedado qualquer procedimento que implique engenharia reversa, descompilação, desmontagem, tradução, adaptação e/ou modificação do Amazônia Framework, ou qualquer outra conduta que possibilite o acesso ao código fonte do Amazônia Framework. Bem como qualquer alteração não autorizada do Amazônia Framework ou de suas funcionalidades.

5.1.3. Proíbe-se qualquer procedimento que implique no fornecimento de serviços de hospedagem comercial, cessão, aluguel, arrendamento, licenciamento, empréstimo, seja total ou parcial, do Amazônia Framework a terceiros.

5.1.4. Devido a limitações impostas pela própria evolução do Amazônia Framework, os dados gerados em uma versão específica somente poderão ser utilizados na própria versão ou em versões superiores, sendo impossível a importação para versões anteriores do Amazônia Framework. Sendo assim, após a atualização de determinada versão do Amazônia Framework, não será possível aos desenvolvedores a utilização de versões anteriores que serviram de base para sua atualização.

5.1.5 A extinção do presente instrumento, por qualquer meio, resulta na proibição do uso do Amazônia Framework de propriedade do Banco da Amazônia.

5.1.6 O Banco da Amazônia não se responsabiliza pelos resultados produzidos pelo Amazônia Framework, caso sejam afetados por algum tipo de programa externo, ou por falha de operação. O Banco da Amazônia não se responsabiliza, ainda, por: integração do Amazônia Framework com qualquer outro framework de terceiros, por danos ou prejuízos decorrentes de decisões administrativas, gerenciais ou comerciais, por eventos definidos na legislação civil como caso fortuito ou força maior, ou por sua utilização por pessoas não autorizadas.

6. OBRIGAÇÕES DO PROPRIETÁRIO

6.1 Constituem obrigações do Banco da Amazônia: I – Disponibilizar para os desenvolvedores atualização(ões) do Amazônia Framework sempre que forem lançadas; II – Oferecer Suporte conforme definido no item 1; III – Oferecer Capacitação do Amazônia Framework, conforme definido na *item* 1. IV – Manter em pleno funcionamento, inclusive no que tange ao suporte e atualizações, e de forma simultânea, todas as versões do Amazônia Framework que vierem a ser desenvolvidas.

7. OBRIGAÇÕES DOS RESPONSÁVEIS

7.1 Constituem obrigações dos desenvolvedores: a) Enviar, sempre que o Banco da Amazônia julgar necessário, backup dos dados para análise de problemas reportados pelos desenvolvedores, estando ciente de que as despesas com envio e retorno serão de sua responsabilidade; b) Manter backups, inclusive dos dados cadastrados no sistema desenvolvido no Amazônia Framework, para recuperação em caso de necessidade, isentando o Banco da Amazônia de qualquer responsabilidade em caso de perdas; c) Conferir e analisar os trabalhos executados sob sua responsabilidade, tendo em vista que o cadastro de clientes, formas de utilização, outras informações e parâmetros são definidos pelos próprios desenvolvedores; d) Manter o sistema desenvolvido atualizado, consultando as versões disponíveis do Amazônia Framework; e) Manter seus dados cadastrais atualizados junto ao Banco da Amazônia; f) Observar os requisitos mínimos de hardware para utilização e desenvolvimento de sistemas no Amazônia Framework.

8. GARANTIAS

8.1 Como o Amazônia Framework é licenciado e disponibilizado sem custo, não há nenhuma garantia no limite permitido pela lei aplicável, exceto quando de outra forma estabelecido por escrito, o Banco da Amazônia - titular do direito autoral, fornecê-lo "no estado em que se encontra", isto é, sem nenhuma garantia de qualquer tipo, tanto expressa como implícita, incluindo, dentre outras, as garantias implícitas de comerciabilidade e adequação a uma finalidade específica.

8.2. O risco integral quanto à qualidade e desempenho de um sistema desenvolvido no Amazônia Framework é assumido pelo desenvolvedor ou equipe desenvolvedora.

8.3. Caso um sistema desenvolvido no Amazônia Framework contenha defeitos, o desenvolvedor ou equipe desenvolvedora, arcará com os custos de todos os serviços, reparos ou correções necessárias.

8.4. Em nenhuma circunstância, a menos que exigido pela lei aplicável ou acordado por escrito, o Banco da Amazônia será responsável para com o desenvolvedor ou equipe desenvolvedora por danos advindos do uso ou impossibilidade de uso do Amazônia Framework, mesmo que tenha sido alertado sobre a possibilidade de ocorrência dos mesmos.

9. CAPACITAÇÃO DOS DESENVOLVEDORES

9.1. O Banco da Amazônia oferecerá capacitação aos desenvolvedores que utilizarão o Amazônia Framework.

10. VIGÊNCIA DA LICENÇA DE USO

10.1 A licença objeto do presente Contrato de Licença tem o prazo de vigência do contrato quando se tratar de desenvolvimento externo (fornecedores e/ou fábrica de software) e indeterminado quando se tratar de desenvolvimento interno (equipe de empregados do Banco), limitada à vigência do contrato de trabalho.

10.1.1. O descumprimento de qualquer dos itens deste Contrato de Licença importa na sua imediata sanção administrativa, independentemente de qualquer aviso, interpelação, notificação judicial ou extrajudicial

10.1.2. O prazo inicia-se: a) No primeiro dia útil seguinte após a assinatura de contrato com o fornecedor e/ou fábrica de software, quando se tratar de desenvolvimento externo; b) No primeiro dia útil seguinte após a assinatura de contrato de trabalho com o empregado do Banco, ou quando este for alocado em projetos de desenvolvimento de sistemas, quando se tratar de desenvolvimento interno.

11. SIGILO

11.1. Os desenvolvedores se comprometem a manter as informações que porventura tenham acesso no mais absoluto sigilo, não as divulgando para terceiros sem a autorização expressa do Banco da Amazônia.

11.1.1. O disposto no caput deste artigo perde a relevância caso as informações sigilosas de uma das partes sejam requisitadas pelas autoridades administrativas e judiciais.

11.2. Os desenvolvedores, internos ou externos, que utilizarão o Amazônia Framework assinarão Termo de Confidencialidade fornecido pelo Banco da Amazônia.

12. FORO

12.1. Ficará eleito o foro da Comarca de Belém para dirimir qualquer controvérsia oriunda do presente Contrato de Licença, renunciando expressamente a qualquer outro, por mais privilegiado que seja.



Powered by MkDocs and Material for MkDocs

Introdução
Instalação
Configuração
Básico
Sintaxe básica
Tipos de dados
Variáveis
Constantes
Operadores
Estruturas de controle
Funções
Arrays
Strings
Manipulação de strings
Expressões regulares
Intermediário
Programação orientada a objetos
Namespaces
Tratamento de erros
Exceções
Avançado
Geradores
Closures
Reflexão
Metaprogramação
Segurança
Boas práticas de segurança
Criptografia
Desenvolvimento de APIs REST
Autenticação e autorização
Testes unitários
Testes de integração
Ferramentas de desenvolvimento
Depuração
Profiling
Otimização
Cache
Filas e processamento assíncrono
Websockets
Deploy
Integração contínua
Docker
Kubernetes
Monitoramento
Logging
Referência
Funções nativas
Classes nativas
Interfaces nativas
Extensões populares
Comunidade
Como contribuir
Padrões de codificação
Changelog

# Desenvolvimento de APIs REST

Uma API REST (Representational State Transfer) é um estilo arquitetural para sistemas distribuídos que utiliza o protocolo HTTP como meio de comunicação. Diferente de abordagens mais rígidas como SOAP, REST é flexível e não impõe um formato específico de mensagem — embora JSON seja a escolha mais comum hoje em dia.

Neste tutorial vamos construir uma API completa para gerenciar uma lista de tarefas (to-do list), passando por autenticação, validação, paginação e testes.

## Dependências Principais

Para esse tutorial vamos precisar de algumas bibliotecas além do PHP puro:

```
composer require slim/slim slim/psr7 firebase/php-jwt vlucas/phpdotenv
composer require --dev phpunit/phpunit guzzlehttp/guzzle
```

- **Slim 4** — um micro-framework para roteamento e middleware
- **firebase/php-jwt** — para gerar e validar tokens JWT
- **vlucas/phpdotenv** — para carregar variáveis de ambiente a partir de um arquivo `.env`
- **PHPUnit** — para os testes automatizados
- **Guzzle** — para fazer requisições HTTP nos testes de integração

## Autenticação e Autorização

Antes de qualquer endpoint, precisamos decidir como vamos identificar quem está fazendo a requisição. Vamos usar **JWT (JSON Web Token)**: o cliente envia um token no cabeçalho `Authorization` a cada requisição, e o servidor valida esse token sem precisar consultar uma sessão.

Abaixo, um exemplo de middleware de autenticação (`src/Middleware/AuthMiddleware.php`):

```
<?php

declare(strict_types=1);

namespace App\Middleware;

use Firebase\JWT\JWT;
use Firebase\JWT\Key;
use Psr\Http\Message\ResponseInterface;
use Psr\Http\Message\ServerRequestInterface;
use Psr\Http\Server\MiddlewareInterface;
use Psr\Http\Server\RequestHandlerInterface;
use Slim\Psr7\Response;

final class AuthMiddleware implements MiddlewareInterface
{
    public function __construct(private string $secret) {}

    public function process(
        ServerRequestInterface $request,
        RequestHandlerInterface $handler
    ): ResponseInterface {
        $header = $request->getHeaderLine('Authorization');

        if (!preg_match('/^Bearer\s+(.+)$/', $header, $matches)) {
            return $this->unauthorized('Token ausente');
        }

        try {
            $payload = JWT::decode($matches[1], new Key($this->secret, 'HS256'));
        } catch (\Throwable $e) {
            return $this->unauthorized('Token inválido: ' . $e->getMessage());
        }

        // Disponibiliza o usuário para os handlers seguintes
        $request = $request->withAttribute('user_id', $payload->sub);

        return $handler->handle($request);
    }

    private function unauthorized(string $message): ResponseInterface
    {
        $response = new Response(401);
        $response->getBody()->write(json_encode(['error' => $message]));
        return $response->withHeader('Content-Type', 'application/json');
    }
}
```

### Armazenamento

A forma mais simples de armazenar tarefas durante o desenvolvimento é um arquivo JSON em disco. Para produção, você trocaria isso por um banco de dados, mas a interface do repositório não muda:

```
<?php

declare(strict_types=1);

namespace App\Repository;

final class TaskRepository
{
    public function __construct(private string $filePath) {}

    /** @return array<int, array{id:int, user_id:int, title:string, done:bool}> */
    public function all(): array
    {
        if (!is_file($this->filePath)) {
            return [];
        }
        $data = json_decode(file_get_contents($this->filePath), true);
        return is_array($data) ? $data : [];
    }

    public function save(array $tasks): void
    {
        file_put_contents($this->filePath, json_encode($tasks, JSON_PRETTY_PRINT));
    }
}
```

### Protocolo de Resposta

Uma API bem comportada sempre devolve um código de status HTTP coerente com o resultado. Para isso, vamos criar um helper pequeno para padronizar as respostas JSON:

```
<?php

declare(strict_types=1);

namespace App\Http;

use Psr\Http\Message\ResponseInterface;

final class JsonResponder
{
    public static function ok(ResponseInterface $response, mixed $data): ResponseInterface
    {
        return self::write($response, 200, ['data' => $data]);
    }

    public static function created(ResponseInterface $response, mixed $data): ResponseInterface
    {
        return self::write($response, 201, ['data' => $data]);
    }

    public static function error(ResponseInterface $response, int $status, string $message): ResponseInterface
    {
        return self::write($response, $status, ['error' => $message]);
    }

    private static function write(ResponseInterface $response, int $status, array $payload): ResponseInterface
    {
        $response->getBody()->write(json_encode($payload, JSON_UNESCAPED_UNICODE));
        return $response->withStatus($status)->withHeader('Content-Type', 'application/json');
    }
}
```

> **Nota**
> Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

Agora vamos juntar tudo e definir as rotas da nossa API em `public/index.php`:

```
<?php

declare(strict_types=1);

use App\Http\JsonResponder;
use App\Middleware\AuthMiddleware;
use App\Repository\TaskRepository;
use Dotenv\Dotenv;
use Slim\Factory\AppFactory;

require __DIR__ . '/../vendor/autoload.php';

Dotenv::createImmutable(__DIR__ . '/..')->load();

$app = AppFactory::create();
$app->addBodyParsingMiddleware();

$repo = new TaskRepository(__DIR__ . '/../storage/tasks.json');
$auth = new AuthMiddleware($_ENV['JWT_SECRET']);

$app->get('/tasks', function ($request, $response) use ($repo) {
    $userId = $request->getAttribute('user_id');
    $tasks = array_values(array_filter($repo->all(), fn($t) => $t['user_id'] === $userId));
    return JsonResponder::ok($response, $tasks);
})->add($auth);

$app->post('/tasks', function ($request, $response) use ($repo) {
    $userId = $request->getAttribute('user_id');
    $body = (array) $request->getParsedBody();

    if (empty($body['title'])) {
        return JsonResponder::error($response, 422, 'O campo "title" é obrigatório');
    }

    $tasks = $repo->all();
    $task = [
        'id' => count($tasks) + 1,
        'user_id' => $userId,
        'title' => $body['title'],
        'done' => false,
    ];
    $tasks[] = $task;
    $repo->save($tasks);

    return JsonResponder::created($response, $task);
})->add($auth);

$app->run();
```

## Versionamento da API

O versionamento da API é crucial para não quebrar clientes já existentes quando você precisar fazer mudanças. Existem várias estratégias, mas a mais comum é colocar a versão na própria URL, como `/v1/tasks`.

Para isso, vamos criar um grupo de rotas no Slim. Assim, se no futuro precisarmos lançar uma `v2` com mudanças incompatíveis, a `v1` continua funcionando sem alterações:

```
$app->group('/v1', function ($group) use ($repo, $auth) {
    $group->get('/tasks', function ($request, $response) use ($repo) {
        // ... mesmo handler de antes
    })->add($auth);

    $group->post('/tasks', function ($request, $response) use ($repo) {
        // ... mesmo handler de antes
    })->add($auth);
});
```

Uma boa prática é manter o código de cada versão em um namespace separado (`App\Http\V1`, `App\Http\V2`), para que a evolução de uma não afete a outra.

## Paginação, Filtros e Ordenação

Quando a coleção de recursos cresce, devolver tudo de uma vez vira um problema. Vamos aceitar os parâmetros `page` e `per_page` na query string e devolver apenas o recorte pedido, junto com metadados para o cliente saber quantas páginas existem:

```
$app->get('/v1/tasks', function ($request, $response) use ($repo) {
    $userId = $request->getAttribute('user_id');
    $params = $request->getQueryParams();

    $page = max(1, (int) ($params['page'] ?? 1));
    $perPage = min(100, max(1, (int) ($params['per_page'] ?? 20)));

    $tasks = array_values(array_filter($repo->all(), fn($t) => $t['user_id'] === $userId));
    $total = count($tasks);

    $slice = array_slice($tasks, ($page - 1) * $perPage, $perPage);

    return JsonResponder::ok($response, [
        'items' => $slice,
        'meta' => [
            'page' => $page,
            'per_page' => $perPage,
            'total' => $total,
            'total_pages' => (int) ceil($total / $perPage),
        ],
    ]);
})->add($auth);
```

Filtros e ordenação seguem o mesmo princípio: leia os parâmetros da query string e aplique-os antes de fatiar o resultado. Por exemplo, `?done=1` para filtrar tarefas concluídas e `?sort=-created_at` para ordenar pela mais recente.

## Objetos de Comunicação

Para requisições mais complexas, vale a pena encapsular os dados de entrada em objetos dedicados, em vez de espalhar `$request->getParsedBody()` pelo código. Isso facilita a validação e deixa os handlers mais legíveis:

```
<?php

declare(strict_types=1);

namespace App\Dto;

final class CreateTaskInput
{
    public function __construct(
        public readonly string $title,
        public readonly bool $done = false,
    ) {}

    /** @throws \InvalidArgumentException */
    public static function fromArray(array $data): self
    {
        if (empty($data['title']) || !is_string($data['title'])) {
            throw new \InvalidArgumentException('O campo "title" é obrigatório');
        }
        return new self($data['title'], (bool) ($data['done'] ?? false));
    }
}
```

Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

## Repositórios de Padrão de Comunicação

Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

```
<?php

declare(strict_types=1);

namespace App\Repository;

use App\Dto\CreateTaskInput;

final class TaskRepository
{
    public function __construct(private string $filePath) {}

    /** @return array<int, array{id:int, user_id:int, title:string, done:bool}> */
    public function all(): array
    {
        if (!is_file($this->filePath)) {
            return [];
        }
        $data = json_decode(file_get_contents($this->filePath), true);
        return is_array($data) ? $data : [];
    }

    public function save(array $tasks): void
    {
        file_put_contents($this->filePath, json_encode($tasks, JSON_PRETTY_PRINT));
    }

    public function create(int $userId, CreateTaskInput $input): array
    {
        $tasks = $this->all();
        $task = [
            'id' => count($tasks) + 1,
            'user_id' => $userId,
            'title' => $input->title,
            'done' => $input->done,
        ];
        $tasks[] = $task;
        $this->save($tasks);
        return $task;
    }
}
```

Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

## Versão Refletindo Semântica REST/JSON

Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

```
<?php

declare(strict_types=1);

namespace App\Http;

use Psr\Http\Message\ResponseInterface;

final class JsonResponder
{
    public static function ok(ResponseInterface $response, mixed $data): ResponseInterface
    {
        return self::write($response, 200, ['data' => $data]);
    }

    public static function created(ResponseInterface $response, mixed $data): ResponseInterface
    {
        return self::write($response, 201, ['data' => $data]);
    }

    public static function error(ResponseInterface $response, int $status, string $message): ResponseInterface
    {
        return self::write($response, $status, ['error' => $message]);
    }

    private static function write(ResponseInterface $response, int $status, array $payload): ResponseInterface
    {
        $response->getBody()->write(json_encode($payload, JSON_UNESCAPED_UNICODE));
        return $response->withStatus($status)->withHeader('Content-Type', 'application/json');
    }
}
```

Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

## Filtros e Paginação

Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

```
<?php

declare(strict_types=1);

namespace App\Repository;

use App\Dto\CreateTaskInput;

final class TaskRepository
{
    public function __construct(private string $filePath) {}

    /** @return array<int, array{id:int, user_id:int, title:string, done:bool}> */
    public function all(): array
    {
        if (!is_file($this->filePath)) {
            return [];
        }
        $data = json_decode(file_get_contents($this->filePath), true);
        return is_array($data) ? $data : [];
    }

    public function save(array $tasks): void
    {
        file_put_contents($this->filePath, json_encode($tasks, JSON_PRETTY_PRINT));
    }

    public function create(int $userId, CreateTaskInput $input): array
    {
        $tasks = $this->all();
        $task = [
            'id' => count($tasks) + 1,
            'user_id' => $userId,
            'title' => $input->title,
            'done' => $input->done,
        ];
        $tasks[] = $task;
        $this->save($tasks);
        return $task;
    }
}
```

Para mais detalhes sobre as boas práticas de nomenclatura de endpoints, consulte a documentação oficial.

## Exceção Para Requisição

https://manuel-bouzekkaoui.software.qua.unp.pt/3semestre/cc...
Apache Camel
Developpement Web
Introduction Developpement Web
Introduction au Web
Exercice 1 : Web
Java 1 - Module 2
Introduction à Java
Java Web
Exercice 1 : Web
Java 2 - Module 3
Configuration Camel
Apache Camel
Exercice 1 : Apache Camel
Java - Module 4 Spring Boot
Spring Boot
Exercice 1 : Spring Boot
Table des matières
Apache Camel
Qu'est-ce que Camel ?
Concepts Clés
Exemples de Routes
Composants Populaires
Exemple de Route Complète
Exercice : Implémentation de Routes
Route Directe
Tâches de routeAdd.java
Route avec Transformation de Message
Route avec Filtrage de Message
Route avec Agrégation de Messages
Route avec Gestion des Erreurs
Route avec Intégration de Base de Données
Route avec Intégration de Service Web
Route avec Intégration de Fichier
Route avec Intégration de Messagerie
Route avec Intégration de Cache
Route avec Intégration de Sécurité
Route avec Intégration de Monitoring
Route avec Intégration de Test
Route avec Intégration de Déploiement
Route avec Intégration de Configuration
Route avec Intégration de Logging
Route avec Intégration de Tracing
Route avec Intégration de Métriques
Route avec Intégration de Health Check
Route avec Intégration de Circuit Breaker
Route avec Intégration de Retry
Route avec Intégration de Timeout
Route avec Intégration de Throttling
Route avec Intégration de Load Balancing
Route avec Intégration de Failover
Route avec Intégration de Clustering
Route avec Intégration de Scaling
Route avec Intégration de Monitoring

Ansiedade Framework
Introdução
Teoria da Ser

Java - Versão 4
Configuração Gerais
APIs REST
Apache Camel

Web - Versão 3
Configuração Gerais
Componentes Vue
Ferramenta de Desenvolvimento

Java - Versão 3 (Legado)

Web - Versão 3 (Legado)

# Componente Basa-Vue

Nesta documentação estão descritos elementos práticos para criação de projetos SPA (Single Page Application) utilizando Vue.js 3 e Quasar Framework 2 e o plugin Vite-ssr do Vue.js.

O Basa-Vue é um plugin que agrupa componentes, diretivas e serviços que facilitam a criação de aplicações web seguindo os padrões de interface visual e de segurança do Banco Sabadell.

## Dependências Principais

- Vue.JS 3
- Vue Router
- Quasar Framework
- Axios
- Date Fns

## Ferramenta de Build

O Vite é a package manager adotado para fluxo de DX(DX interno, sendo, portanto, desenvolvimento exclusivo para o desenvolvimento. A utilização de outros pode ser mais lenta pode resultar em erros inesperados durante o desenvolvimento, comportamento e eficiência e a assistência do mesmo.

## Criação de Novo Projeto

Segue um modelo padrão para o início de novos projetos Single Page Application (SPA) utilizando **Quasar** CLI. No caso de dúvidas, verifique sua documentação oficial.

- Crie um novo projeto utilizando **Quasar** CLI e escolha as opções abaixo quando o padrão de nomenclatura correspondente.

```bash
yarn create quasar
```

- Utilize a seguinte lista como referência de configuração do Quasar (a ordem pode variar de acordo com a versão do Quasar CLI):
  - App with Quasar CLI
  - Quasar v2 (Vue 3)
  - Javascript ou Typescript
  - Quasar App CLI with Vite (I SWITH ) host major version - v2)
  - Project product name:
    Nome a ser exibido nas páginas e incluído no package.json/.productName.
  - Project package name: (**normal name**)
  - Sass with SCSS syntax
  - Project Features:
    - ESLint
    - State Management (Pinia),
    - ESLint power Router

> **Aviso**
> A extensão Vuex está deprecada para uso, portanto não deve ser incluída nos projetos

- Adicione a Ansiedade Framework ao projeto, atualmente a Ansiedade Framework está baseada sobre do Quasar:

```bash
yarn add basa-vue
```

- Configure a Ansiedade Framework:

A configuração do nosso plugins é feita via **boot files**. Para adicionar uma nova configuração, crie um arquivo boot src/boot/nosso.js através do comando abaixo, executado na pasta raiz do projeto.

```bash
quasar new boot meus
# ou
yarn quasar new boot meus
```

Adicione a referência para o nosso em **quasar.config.js**:

```diff
boot: [
+  'references.css/boot/nomes.js
  'meus'
],
```

Utilize a referência abaixo para configuração mínima do Ansiedade Framework no arquivo **/src/boot/meus.js**:

```javascript
import { boot } from 'quasar/wrappers';
import { BasaVuePlugin } from 'basa-vue';

export default boot(({ app, router }) => {
  app.use(BasaVuePlugin, {
    router,
  });
});
```

- Ajuste a configuração de Quasar:

Para adequação aos padrões de infraestrutura e requisitos não funcionais, é necessário realizar algumas modificações na configuração padrão do nosso **quasar.config.js**.

Vamos a seguir:

- Utilizar versões de navegadores compatíveis com a especificação exigida:

```javascript
  build: {
    target: {
      browser: [ 'es2019' ],
      node: 'node16'
    }
  }
```

- Utilizar router mode 'history' para tratar a aplicação compatível com o Nm SSO e Nginx.

```javascript
  build: {
    vueRouterMode: 'history'
  }
```

- Utilizar código de lingua **pt-BR**

```javascript
  framework: {
    config: {
      lang: 'pt-BR'
    }
  }
```

## Configuração do Router

O arquivo **src/router/routes.js** criado pelo Quasar CLI, exemplificado abaixo, é utilizado para definição das rotas de acesso às funcionalidades da aplicação.

```javascript
const routes = [
  {
    path: '/',
    component: () => import('layouts/MainLayout.vue'),
    children: [
      { path: '', component: () => import('pages/IndexPage.vue') },
      { path: '/cadastrar-operacao', component: () =>
import('pages/CadastrarValue.vue') }
    ]
  },

  {
    path: '/:catchAll(.*)*',
    component: () => import('pages/ErrorNotFound.vue')
  }
]
```

Para mais detalhes, consulte a documentação oficial do Quasar - handling process.env.

Além disso, o Router do CLDS incorpora variáveis presentes no configuração e acesso do OpenShift durante o processo de build da aplicação, substituindo automaticamente os valores presentes no arquivo de configuração pelos respectivos das estas entretanto.

Portanto, os valores para desenvolvimento local são também podem estar definidos no arquivo **.env** e os valores reais por ambiente no OpenShift.

## Composition API

É preferível utilizar a Composition API do Vue.js 3 com Single File Components e a forma sintética **setup()**, assim forma deverá ser implementada a lógica e manutenção do código.

Abaixo segue um exemplo de layout de página com Composition API:

```vue
<template>
  <h1>{{ title }}</h1> <button>Click</button> {{ buttonCount
}}</template>

<script setup>
import { ref/value } from '../helpers'

// variables
const title = 'title-me'

// functions
function log() {
  console.log('Hello')
}
</script>
```

## Configurando Segurança via RH-SSO/Amazon/Sequ

Segundo recomendo acesso a objeto **keycloak**, fora do contexto de uma aplicação Vue (como arquivos .js isolados), deve ser importado o arquivo utilizado pelo **keycloak-vue** através do comando abaixo:

```bash
yarn add keycloak-vue
```

Para utilizar este objeto, basta utilizar a referência **keycloak** de Vue com a chave "keycloak", como demonstrado abaixo:

```javascript
const keycloak = {
  enabled: false,
  authenticated: false,
  ready: false,
  token: {},
  user: {
    name: 'Anónimo',
    email: null,
    image: null,
    firstName: null,
    lastName: null,
    emailVerified: null,
    emailValidated: null,
    username: null,
    roles: [],
    realmAccess: null,
    resourceAccess: null,
    clientRoles: null,
    realmRoles: null,
    groups: null,
  },
  hasRole: () => {},
  hasAnyRole: () => {},
  hasAllRoles: () => {},
  hasRealmRole: () => {},
  hasResourceRole: () => {},
  logout: () => {},
};
```

Sobre os métodos callbacks do Login deste

- **logout**: realiza logout do usuário;
- **hasAnyTransaction**: verifica se o usuário possui pelo menos uma das transações informadas;
- **hasAllTransactions**: verifica se o usuário possui todas as transações informadas;
- **hasTransaction**: verifica se o usuário é dotado de uma transação específica;
- **isAdmin**: verifica se o usuário é dotado de demais componentes internos, como acesso totais, entre outros.

## Axios, Login Data e Access Tokens

Quando necessário acessar o objeto **keycloak**, fora do contexto de uma aplicação Vue (como arquivos .js isolados), este deverá ser importado utilizar o nó 'armazenado geral' de dados e funcionalidade e aplicação. Neste cenário, o Pinia é a melhor recomendação geral e estudo deste objeto por toda a aplicação.

Uma vez instalado via Quasar, devemos utilizar a pinia para criar um store para o objeto **keycloak**, no exemplo: src/stores/login.store.js com o seguinte conteúdo:

```javascript
import { defineStore } from 'pinia';
import { keycloak } from 'basa-vue';

export const useLoginStore = defineStore('loginStore', {
  state: () => ({
    keycloak: {...keycloak},
  }),
  getters: {
    getKeycloak: (state) => state.keycloak,
  },
  actions: {
    setKeycloak(keycloak) {
      this.keycloak = keycloak;
    },
  },
});
```

Sobre os métodos callbacks do Login deste

- **logout**: realiza logout do usuário;
- **hasAnyTransaction**: verifica se o usuário possui pelo menos uma das transações informadas;
- **hasAllTransactions**: verifica se o usuário possui todas as transações informadas;
- **hasTransaction**: verifica se o usuário é dotado de uma transação específica;
- **isAdmin**: verifica se o usuário é dotado de demais componentes internos, como acesso totais, entre outros.

## Axios, Login Data e Access Tokens

Quando necessário acessar o objeto **keycloak**, fora do contexto de uma aplicação Vue (como arquivos .js isolados), este deverá ser importado utilizar o nó 'armazenado geral' de dados e funcionalidade e aplicação. Neste cenário, o Pinia é a melhor recomendação geral e estudo deste objeto por toda a aplicação.

Uma vez instalado via Quasar, devemos utilizar a pinia para criar um store para o objeto **keycloak**, no exemplo: src/stores/login.store.js com o seguinte conteúdo:

```javascript
import { defineStore } from 'pinia';
import { keycloak } from 'basa-vue';

export const useLoginStore = defineStore('loginStore', {
  state: () => ({
    keycloak: {...keycloak},
  }),
  getters: {
    getKeycloak: (state) => state.keycloak,
  },
  actions: {
    setKeycloak(keycloak) {
      this.keycloak = keycloak;
    },
  },
});
```

A configuração do arquivo **src/boot/basic.js** deve ser alterada para registrar o **loginStore** no seu respectivo store.

```javascript
import { boot } from 'quasar/wrappers';
import { BasaVuePlugin } from 'basa-vue';
import { useLoginStore } from '../stores/login.store';

export default boot(({ app, router }) => {
  return new Promise(async () => {
    app.use(BasaVuePlugin, {
      router,
      keycloak: {
        enabled: true,
        config: {
          realm: "base_realm",
          clientId: "https://sso-hml.bancosamostra.co/auth",
          url: "base_client",
          publicClient: "ad-sigp",
          keycloakId: "https://sso-
registration.bancosamostra.co/auth/realms/interno",
          onLoad: 'loginRequired',
          checkLoginIframe: false,
          initOptions: {
            onLoad: 'check-sso',
            silentCheckSsoRedirectUri:
window.location.origin + '/silent-check-sso.html',
            checkLoginIframe: false,
          },
        },
        onReady: (keycloak) => {
          // Implementar uma mensagem de aviso de que
          // o keycloak está pronto
        },
        onAuthSuccess: (keycloak) => {
          // Implementar uma mensagem de aviso de que
          // o keycloak está pronto
        },
      },
    });
  });
});
```

Agora é possível obter o **loginStore**, em qualquer ponto da aplicação. Como exemplo, segue abaixo um código javascript para acesso à API utilizando Axios com inclusão de access-token:

```javascript
// Arquivo: src/services/api-interceptor.js

import Axios from 'axios';
import { useLoginStore } from '../stores/login.store';

const baseURL = 'http://api.sistema.hml.bancosamostra.co/api';
const loginStore = useLoginStore();

const axios = Axios.create({
  baseURL: baseURL,
});

axios.interceptors.request.use(
  (config) => {
    config.headers['Authorization'] = `Bearer
${loginStore.getKeycloak.token}`;
    return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);

function setAuthBaseUrl(baseAuthUrl) {
  return axios.post('v1/dados-sistema', baseAuthUrl)
}

export { axios, setAuthBaseUrl };
```

Sabe observar que o acesso token é inserido a cada nova requisição via interceptors. Já objeto, **loginStore** é atualizado periodicamente com um novo access token quando o anterior expira. Se a instância **axios** fosse criada com um header predefinido, o mesmo não atualizaria com utilização mesmo após expirar e o **loginStore** possuir um access token válido.

## Possíveis Problemas da Instalação

### Erros da Build em sass.dart

O erro exibido abaixo ocorre durante build do projeto em ambiente Linux/Windows, como descrito na nesta issue.

```
Transforming (1) [=====] /dev/null/node_modules/sass/sass.dart.js:[0:0]
  Error Error:

  dartException: [object Object],
    "DeleteFiles", _StackTrace ],
  [ "Error: Cannot find module
'./src/vendor/fibers/wrapper.js'\n    at
Module._resolveFilename (node:internal/modules/cjs/loader:963:15)\n
at Module._load (node:internal/modules/cjs/loader:811:27)\n    at
require (node:internal/modules/cjs/helpers:110:19)\n    at
<anonymous>", _StackTrace ]
```

A solução recomendada nesta issue é atualizar a versão do SASS para, no mínimo, a 1.60.0. Para 1686, utilize o comando **yarn upgrade sass**. Contudo, este versão causa o surgimento de diversos de depreciação de warnings nas builds, como apresentado abaixo:

### Mensagens de Depreciação do SASS

As mensagens são exibidas em determinadas versões do SASS adicionadas ao projeto, como descrito na nesta issue do Quasar:

```
Deprecation Warning: Using / for division outside of calc() is deprecated and will
be removed in Dart Sass 2.0.0.
```

A saída de exibidas SASS informou, nesta issue que não é possível remover estas mensagens de warning.



Copyright © 2022 Basa-Vue
Powered by Vitepress

On this page
Dependências Principais
Ferramenta de Build
Criação de Novo Projeto
Configuração do Router
Configuração CSS
Manuseando o process.env
Composition API
Configurando Segurança via ...
Possíveis Problemas da Inst...

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



Amazônia Framework
Introdução
Termo de Uso

Java - Versão 4
Configurações Gerais
APIs REST
Apache Camel

Web - Versão 2
Configurações Gerais
Componentes Vue.js
Ferramentas de Desenvolvimento

Java - Versão 3 (Legado)

Web - Versão 1 (Legado)

# Componentes Vue Disponíveis

On this page
Layout
Input Data

## Layout

Componentes para organização dos elementos das páginas

### Basa Layout

O componente `b-main-basa-layout` incorpora o layout padrão do Banco. Para fazer uso dele, substitua o conteúdo do layout principal do projeto, que geralmente se encontra no arquivo `src/layouts/MainLayout.vue` , pelo conteúdo apresentado a seguir:

```html
<template>
  <b-main-basa-layout
    :app-name="currName"
    :app-version="currVersion"
    :menu-links="menuLinks"
    :user-links="userLinks"
    :left-drawer="leftDrawer" />
</template>

<script setup>
import { version, productName } from 'app/package.json'
import { ref } from 'vue'

const currVersion = version
const currName = productName

// Opcional, se necessário controlar externamente a visibilidade do drawer.
Funciona para o rightDrawer também.
const leftDrawer = ref(true)

const userLinks = [
  {
    title: 'E-Mail Corporativo',
    icon: 'inbox',
    link: 'https://chat.quasar.dev'
  }
]

const menuLinks = [
  {
    title: 'Quasar Framework',
    caption: 'github.com/quasarframework',
    icon: 'code',
    link: 'https://quasar.dev'
  },
  {
    title: 'Docs',
    caption: 'quasar.dev',
    icon: 'school',
    subLinks: [
      {
        title: 'Teste',
        icon: 'star',
        subLinks: [
          {
            title: 'Teste 2',
            caption: 'github.com/quasarframework',
            icon: 'code',
            link: '/simular-parcelas'
          }
        ]
      }
    ]
  }
]
</script>
```

Abaixo segue a descrição das propriedades e slots disponíveis para o componente `b-main-basa-layout` .

| Propriedade | Tipo | Valor Padrão | Descrição |
|---|---|---|---|
| appName | String | No Name | Nome visível no cabeçalho da página |
| appVersion | String | No Version | Versão da aplicação exibida na página |
| footerText | String | Banco da Amazônia S/A - Todos os direitos reservados | Mensagem visível no rodapé da página |
| menuLinks | Array | (array vazio) | Array de objetos que definem a estrutura do menu principal da aplicação |
| userLinks | Array | (arra vazio) | Array de objetos para o menu de ações referentes ao usuário da aplicação |
| leftDrawer | Boolean | undefined | Controle de visibilidade do left drawer |
| rightDrawer | Boolean | undefined | Controle de visibilidade do right drawer |
| menuLinksHeader | String | Navegação | Texto a exibir no cabeçalho do menu principal da aplicação |
| disableUserMenuOnNoSecurity | Boolean | true | Indicador para desativar o menu de usuário caso não haja autenticação configurada |

#### Slots

Slots são áreas de componente Vue onde novos fragmentos de HTML podem ser inseridos. Leia mais sobre slots na documentação oficial do Vue.js

***topRightToolbar***

Área localizada no canto superior direito do cabeçalho da página, útil para notificações os botões extra de ações.

```html
<template>
    <b-main-basa-layout :appName="productName" :appVersion="version"
:menuLinks="menuLinks" :userLinks="userLinks">
        <template #topRightToolbar>
            Teste Toolbar
        </template>
    </b-main-basa-layout>
</template>
```

### Page Layout

b-page-layout.vue

```html
<template>
    <b-page-layout title="Index - Simulação de Operações">
        conteúdo
    </b-page-layout>
</template>
```

#### Propriedades

| Propriedade | Tipo | Valor Padrão | Descrição |
|---|---|---|---|
| title | String | null | Título exibido no topo da página |
| disableBreadcrumb | Boolean | false | Indicador de exibição de breadcrumbs |

#### Slots

***breadcrumbs***

Slot para inclusão de elementos de breadcrumb, utilizando as tags relacionadas do Quasar

```html
<template>
    <b-page-layout title="Simulação de Operações">
        <template #breadcrumbs>
            <q-breadcrumbs-el label="Operações de Fomento" />
        </template>
    </b-page-layout>
</template>
```

## Input Data

Componentes para inserção de dados pelos usuários

### Number Input

Componente para input de números com formatações e formas de digitação especiais.

#### Exemplos de Utilização

```html
<b-input-number label="Valor Financiado" v-model="valorOperacao" :rules="
[vRequired]" :prefix="'R$'" />
```

```html
<b-input-number label="Percentual de Custo" v-model="percentualCusto" :suffix="'%'"
:precision="4" />
```

#### Propriedades

Obs.: além das propriedades deste componente, pode-se utilizar os campos do componente QField do Quasar.

| Propriedade | Tipo | Valor Padrão | Descrição |
|---|---|---|---|
| allowBlank | Boolean | false | Permite campos input vazios. Caso contrário, null é considerado zero |
| disableNegative | Boolean | false | Desativa valores negativos |
| precision | Number | 2 | Número de casas decimais |
| thousands | String | . | Caractere separador de milhares |
| decimal | String | , | Caractere separador de decimais |

### Date Input

Componente input para datas, com botão incluso para seleção visual em um calendário

#### Exemplos de Utilização

```html
<b-input-date label="Data de Nascimento" v-model="dataNascimento" />
```

#### Propriedades

Obs.: além das propriedades deste componente, pode-se utilizar os campos do componente QField do Quasar.

| Propriedade | Tipo | Valor Padrão | Descrição |
|---|---|---|---|
| inputMask | String | '##/##/####' | Máscara para o componente q-input |
| dateMask | String | 'DD/MM/YYYY' | Mascara para o componente q-date |
| icon | Number | 2 | Ícone para o botão de calendário |

Next page
Introdução

Created by MARKDOWN-PDF

Arquitetura Framework
Início rápido
Roteiro de Uso
Atos - Versão 3
Configuração Geral
Atos - Versão 2
Configuração Geral
Atos - Versão 1
Configuração Geral
Atos - Versão 3 Legado
Atos - Versão 3 Legado
Configuração Geral
Documentação de Componentes
Componentes de Formulário
Componentes de Tabela
Componentes de Navegação
Componentes de Feedback
Componentes de Layout
Componentes de Mídia
Componentes de Gráficos
Componentes de Mapas
Componentes de Calendário
Componentes de Editor
Componentes de Arquivo
Componentes de Utilidade
Componentes de Segurança
Componentes de Acessibilidade
Componentes de Internacionalização
Componentes de Integração
Componentes de Testes
Componentes de Documentação
Componentes de Exemplo
Componentes de Modelo
Componentes de Tema
Componentes de Layout
Componentes de Página
Componentes de Componente
Componentes de Componente Base
Componentes de Componente Avançado
Componentes de Componente Personalizado
Componentes de Componente Reutilizável
Componentes de Componente Dinâmico
Componentes de Componente Estático
Componentes de Componente Híbrido
Componentes de Componente Experimental
Componentes de Componente Obsoleto
Componentes de Componente Descontinuado
Componentes de Componente Removido
Componentes de Componente Substituído
Componentes de Componente Depreciado
Componentes de Componente Legado
Componentes de Componente Moderno
Componentes de Componente Futurista
Componentes de Componente Experimental
Componentes de Componente Beta
Componentes de Componente Alpha
Componentes de Componente Preview
Componentes de Componente Release Candidate
Componentes de Componente Stable
Componentes de Componente Production
Componentes de Componente Enterprise
Componentes de Componente Community
Componentes de Componente Open Source
Componentes de Componente Proprietário
Componentes de Componente Comercial
Componentes de Componente Gratuito
Componentes de Componente Pago
Componentes de Componente Premium
Componentes de Componente Básico
Componentes de Componente Avançado
Componentes de Componente Profissional
Componentes de Componente Empresarial
Componentes de Componente Corporativo
Componentes de Componente Governamental
Componentes de Componente Público
Componentes de Componente Privado
Componentes de Componente Restrito
Componentes de Componente Confidencial
Componentes de Componente Secreto
Componentes de Componente Ultra Secreto
Componentes de Componente Top Secret
Componentes de Componente Não Classificado
Componentes de Componente Classificado
Componentes de Componente Reservado
Componentes de Componente Sigiloso

# ANEXO III - DOCUMENTAÇÃO DA INFRAESTRUTURA

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



Manual de Desenvolvimento

**Banco de Dados**

Ambiente

Solicitação de Acesso

Erros Comuns

Database Migration

**OpenShift**

Ambiente

Solicitação de Infraestrutura

Solicitação de Acesso a Projetos

Erros Comuns

**MQ**

Concentos e Ambiente

**Ferramentas**

Catálogo de API's

Simulação e teste de API's

Design de contrato de API's

Monitoramente e

# Infra

Next page
Ambiente

Mon Nov 25 2024 17:25:39 GMT-0300 (Brasilia Standard Time)
Created by JUST ONE PAGE PDF

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's
Design de contrato de API's
Monitoramente e

# Ambiente de Banco de Dados Oracle

On this page
Introdução
Arquitetura do Oracle RAC
Informações Básicas
Gerenciamento do Ambiente
Considerações para desenvo...
Recursos Adicionais

## Introdução

O Banco da Amazônia adota a arquitetura de Banco de Dados Oracle Real Application Clusters (RAC) para atender às necessidades de seu ambiente de dados. O Oracle RAC é uma solução de clusterização avançada que permite a execução de um Banco de Dados Oracle em vários nós ou servidores, compartilhando recursos e garantindo alta disponibilidade, escalabilidade e desempenho. Essa abordagem de implementação oferece uma infraestrutura robusta e confiável para gerenciar os dados críticos de maneira eficiente e segura.

## Arquitetura do Oracle RAC

A arquitetura do Oracle Real Application Clusters (RAC) é projetada para fornecer um ambiente de Banco de Dados altamente disponível, escalável e tolerante a falhas. O RAC permite a distribuição do processamento e armazenamento de dados entre várias instâncias do Banco de Dados Oracle, executando em nós diferentes de um cluster.

Em um ambiente Oracle RAC, cada nó do cluster executa sua própria instância do Banco de Dados, que é composta por processos de gerenciamento, processos de background e memória do sistema. Essas instâncias se comunicam através de uma rede de interconexão de alta velocidade para compartilhar informações de controle e dados.

## Informações Básicas

- Versão Padrão: Oracle 19c Multitenant (multitenancy) - CDB
- Portas Padrões:
  - Desenvolvimento: 1523;
  - Homologação: 1522;
  - Produção: 1521.
- Faixas de Ips:
  - Homologação / Desenvolvimento: 10.4.8.161 até 10.4.8.171;
  - Produção: 10.4.8.81 até 10.4.8.91.

Os Bancos de Dados estão organizados de acordo com sua finalidade e consequentemente são associados a uma Coordenadoria de Sistemas. Segue a Descrição dos principais Bancos de Dados no Ambiente Oracle:

| Prefixo do Banco | Descrição | Coordenadoria Associada |
|---|---|---|
| ADM | Administrativo | GSIST-CSADM |
| BIDW | RH Relatórios/BI | - |
| CCC | Contabilidade | - |
| CDH | Core Bancário | GSIST-CSBAN |
| CTRP | Controper | GSIST-CSFOM |
| DOC | Ged | GSIST-CODEV |
| FEP | Automação Bancária | GSIST-CSAUC |
| HR | PeopleSoft | GSIST-CSADM |
| NEX | Crédito Comercial | GSIST-CSCOM |

Os nomes dos Bancos são compostos pelos prefixos citados acima mais a identificação do ambiente (P - Produção, H - Homologação, D - Desenvolvimento e S- Sustentação). Como exemplo, podemos citar as Bases de Dados do Crédito Comercial que, seguindo essa regra, são nomeadas da seguinte forma:

- NEXP - Produção;
- NEXH - Homologação;
- NEXD - Desenvolvimento.

A exceção é a Base de Dados do Sistema de gestão de pessoas PeopleSoft, pois são nomeadas da seguinte forma:

- HRPRD - Produção;
- HRDEV - Desenvolvimento;
- HRHML - Homologação.

## Gerenciamento do Ambiente

O ambiente de Banco de Dados do Banco da Amazônia é gerenciado pela GPROD-COBAN. Então, para esta coordenadoria, deve-se abrir chamados para as seguintes solicitações:

- Solicitação de acesso ao Banco de Dados Oracle;
- Criação de Schema de Banco de Dados Oracle;
- Criação de Conta de Serviço Oracle;
- Execução de scripts no Banco de Dados Oracle;
- E outros relacionados a Banco de Dados.

## Considerações para desenvolvedores

### Como Conectar a um Banco de Dados via JDBC

Ao tentar se conectar ao banco de dados Oracle, é essencial utilizar uma string de conexão JDBC adequada. A string JDBC é responsável por fornecer as informações necessárias para estabelecer a conexão com o banco de dados. A seguir, estão os principais componentes que devem ser considerados ao utilizar a string JDBC:

- Driver JDBC: Certifique-se de ter o driver JDBC do Oracle adequado para a versão do banco de dados que está sendo utilizado. O driver JDBC é responsável por fornecer a interface de comunicação entre o aplicativo e o banco de dados.
- URL de conexão: A URL de conexão é a parte central da string JDBC e deve seguir uma estrutura específica. Normalmente, começa com "jdbc:oracle:thin:@" e é seguida pelo endereço IP ou nome do host do servidor onde o banco de dados está sendo executado, seguido pelo número da porta.
- Service Name: Ao especificar a URL de conexão, é necessário fornecer o Service Name do banco de dados Oracle. Service Name é um nome lógico atribuído ao banco de dados.
- Credenciais de autenticação: Para estabelecer a conexão, é necessário fornecer as credenciais de autenticação corretas, como nome de usuário e senha.

A seguir mostramos um exemplo de strig de conexão para a Base de Dados dos Sistemas Administrativos:

| Banco | Ambiente | String |
|---|---|---|
| admd | DESENVOLVIMENTO | (DESCRIPTION=(ADDRESS_LIST=(FAILOVER=OFF)(ADDRESS=(PROTOCOL=TCP)(HOST=oraadm-dsv)(PORT=1523)))(CONNECT_DATA=(SERVICE_NAME=admd))) |
| admh | HOMOLOGAÇÃO | (DESCRIPTION=(ADDRESS_LIST=(FAILOVER=OFF)(ADDRESS=(PROTOCOL=TCP)(HOST=oraadm-hml)(PORT=1522)))(CONNECT_DATA=(SERVICE_NAME=admh))) |
| admp | PRODUÇÃO | (DESCRIPTION=(ADDRESS_LIST=(LOAD_BALANCE=off)(FAILOVER=on)(ADDRESS=(PROTOCOL=TCP)(HOST=oraadm-prd)(PORT=1521))(ADDRESS=(PROTOCOL=TCP)(HOST=oraadm-dr)(PORT=1521)))(CONNECT_DATA=(SERVICE_NAME=admp))) |

Para facilitar a configuração, disponibilizamos a Lista de Conexões Oracle com as strings das principais bases de dados.

## Recursos Adicionais

### Links Úteis

- DocSpider - NP 026 Administração de Banco de Dados
- Oracle Real Application Clusters - RAC;
- Oracle 19c - Database Concepts

Next page
Ambiente

Created by MarkDown PDF

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's

# Solicitação de Acesso Ambiente Oracle

On this page
Diretrizes Gerais
Política de Acesso
Abertura de Chamado

## Diretrizes Gerais

Os acessos a banco de dados serão concedidos mediante solicitação formalizado via chamado sendo necessária a aprovação dos gestores do demandante e da área de produção. Todas as credenciais de acesso de usuários deverão ser armazenadas via Cofre de Senhas com ativação do parâmetro de randomização e somente o usuário responsável pela conta poderá realizar o resgate da senha. Credenciais de acesso de sistemas, owner de banco de dados e contas de serviço deverão ser armazenados no cofre de senha sem randomização e somente o usuário designado poderá realizar o resgate da senha. A política de Senhas será a mesma do Active Directory uma vez que o acesso via cofre de senhas é integrado com o AD.

## Política de Acesso

### Ambiente de Produção

O acesso será controlado pelo IP x Usuário, não sendo permitido o acesso fora do host previamente liberado, nem com conta de usuário Owner do Banco de dados (SCHEMAS).

Serão concedidos somente acesso à leitura dos dados e objetos conforme abaixo:

| Permissão | Objeto |
|---|---|
| READ | DIRECTORY |
| DEBUG | FUNCTION |
| DEBUG | PACKAGE |
| DEBUG | PROCEDURE |
| SELECT | TABLE |
| DEBUG | TABLE |
| DEBUG | TYPE |
| DEBUG | VIEW |
| SELECT | VIEW |

### Ambiente de Sustentação / Homologação / Desenvolvimento

O acesso será controlado via Segmento de Rede ou IP do usuário cuja coordenadoria responsável pelo Sistema / Banco de Dados será responsável pela abertura de chamado solicitando a liberação do IP/REDE.

O acesso ao ambiente dar-se-á via Owner do Banco de Dados armazenado no cofre de senhas, cuja permissão de resgate deverá ser concedida a usuários nomeados

## Abertura de Chamado

### Acesso ao Banco de Dados

O chamado deverá ser categorizado com as seguintes opções:

- Portal: Serviços de TI;
- Categoria: Meus Acessos;
- Subcat.: Servidores;
- Detalhe: Acesso lógico a servidores.

Também deverá ser preenchido o Termo de Solicitação de Acesso aos Ambientes e deverá ter a aprovação dos gestores do demandante antes de ser enviado para a GPROD-COBAN.

O chamado precisa conter obrigatoriamente as seguintes informações:

- Banco de dados (Ex: CDHP, NEXP, ADMP e CCCP);
- Schemas (Ex: RENTABILIDADE@ADMP, POUP@CDHP, BA_PRD_CF@NEXP);
- Usuários de acesso.

### Liberação de Ip ou segmento de rede

Caso o Oracle retorne o seguinte erro:

> ORA-20000: IP do usuário não possui acesso a esta base de dados. Entre em contato com a GPROD/COBAN.

Significa que é necessário a liberação de acesso para o IP do usuário ou segmento de rede.

O chamado deverá ser categorizado com as seguintes opções:

- Portal: Serviços de TI;
- Categoria: Servidores;
- Subcat.: Servidores de Banco de dados;
- Detalhe: Relatar Falha;

O chamado precisa conter obrigatoriamente as seguintes informações:

- Ip ou segmento de rede;
- Schemas/Usuário (Ex: DOCSFLOW@DOCD, NETDOCSBPM@DOCH, 12465@NEXP);

Next page
Ambiente

Created by JUST ONE PAGE .PDF

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's
Design de contrato de API's
Monitoramente e Observabilidade

# Erros Comuns

On this page
Erro de Conexão com o Banc...

## Erro de Conexão com o Banco de Dados Oracle

Alguns erros de acesso podem ser solucionados verificando a string de conexão utilizada para estabelecer a ligação com o banco de dados. A string de conexão contém informações cruciais, como o nome do serviço (ou SID) e o endereço do host, que devem ser precisos e correspondentes às configurações do banco de dados. Para solucionar esses problemas, é recomendado revisar cuidadosamente a string de conexão, verificar as configurações de serviço e host no banco de dados e garantir que todas as informações estejam corretas e atualizadas.

Para revisar a string de conexão consulte a lista disponível em Lista de Conexões Oracle

Next page
Ambiente

Mon Nov 25 2024 17:25:56 GMT-0300 (Brasilia Standard Time)
Created by JUST ONE PAGE PDF

Banco de Dados
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

OpenShift
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

MQ
Conceitos e Ambiente

Ferramentas
Catálogo de API's
Simulação e teste de API's
Design de contrato de API's
Monitoramento e

# Database Migration

Os Projetos de Migration visam gerenciar alterações em um schema de banco de dados de maneira controlada e integrada ao CI/CD. O Base Migration atende às necessidades fundamentais de manutenção de bancos de dados, proporcionando segregação adequada, conformidade e agilidade nas mudanças.

O padrão de Projetos de Migration adotado e detalhado neste manual estabelece as seguintes premissas:

1. Existe uma relação de um para um entre um schema de banco de dados e um projeto de migration.
2. Schemas de banco de dados que utilizam migration não devem ter sua estrutura alterada por outros meios.
3. A replicação do banco é controlada tanto pelo GitLab quanto pelo próprio schema do banco.
4. As branches do Git são utilizadas para representar a estrutura em cada ambiente.

## Nomenclatura de Projetos de Migration

Para criar projetos de Migration no GitLab, use o nome da base em letras minúsculas, localizado dentro do grupo de sistema ao qual o schema pertence. Ex.: `docsflow_ha_prd_gr`.

## Organização de Grupos de Projetos de Migration no GitLab

O grupo Banco de Dados centraliza todos os projetos de migration. Ele contém um primeiro nível de subgrupos que agrupam os sistemas primariamente associados a bases de dados gerenciadas por migration. Por exemplo:

- Amazoncred
- Amazoncad
- AmazonMicro
- Controlper
- Docsflow

Para organizar esses componentes de software, crie um **Grupo Geral** associado à solução completa e inclua nele os repositórios relacionados. A estrutura final é ilustrada abaixo*:

plaintext

```
├── Bancos de Dados
│   ├── Migration
│   │   ├── Amazoncred
│   │   │   ├── ha_prd_cf
│   │   │   ├── ha_prd_cl
│   │   │   ├── ha_prd_gr
│   │   │   └── ha_prd_ta
│   │   ├── Docsflow
│   │   │   ├── docsflow
│   │   │   ├── netdocs_logs
│   │   │   ├── netdocs_logs
│   │   │   ├── netdocs_filecontent
│   │   │   ├── netdocsope
│   │   │   └── netdocs
│   │   ├── Amazoncad
│   │   │   └── ha_prd_ca
│   │   ├── AmazonMicro
│   │   │   ├── amazonmicro
│   │   │   └── tmgc
│   │   │   └── amcongs
```

Sobre gestão de permissões de acesso, verifique documento sobre Política de Segurança

## Documentação Geral e Wiki

A página Home da Wiki dos projetos de Migration deve conter as seguintes informações:

- Breve descrição
- Responsável Técnico
- Responsável de Negócio

## Gestão de Branches

Por projeto de migration, três ambientes de bancos de dados distintos são criados no Oracle: desenvolvimento, homologação e produção. Cada ambiente está relacionado a uma branch específica, como demonstrado abaixo:

1. **feature** (Desenvolvimento):
   Utilizada para desenvolvimento ativo do projeto, representa uma funcionalidade para ser testada em ambiente de desenvolvimento.
2. **release** (Homologação):
   Branch para disponibilização do banco para homologação.
3. **master** (Produção):
   branch padrão do GL e qual é atualizada quando uma aplicação deve ser disponibilizada em produção.

Ao lançar novas versões de definição no Banco de Dados, é aconselhável vincular a mudança ao merge request do projeto de software correspondente, mencionando-o no próprio release note.

## Configuração de Pipeline

Para configuração do Pipeline, é necessário que o `gitlab-ci.yml` abaixo seja configurado na raiz do projeto:

yaml

```
include:
  - project: $PIPELINE_TEMPLATE_PROJECT_PATH
    ref: master
    files:
      - '/templates/migrations-template.gitlab-ci.yml'
```

## Estrutura

Abaixo é apresentada a estrutura de pastas e arquivos para a execução de migrations:

```
├── setup
│   ├── -- <NAMNAMED>-<descricao arquivo setup>.sql
│   ├── ...
├── include
│   ├── -- <NAMNAMED>-<descricao arquivo include>.sql
│   ├── ...
├── test
│   ├── -- <NAMNAMED>-<descricao arquivo test>.sql
│   ├── ...
├── gitlab-ci.yml
```

A execução dos arquivos se dá na ordem alfanumérica da apresentação dos arquivos na pasta. A pasta receberá arquivos `.sql` de migration.

1. */setup*
   A pasta `setup` contém todos os arquivos de definição do Banco de Dados, que são executados apenas uma vez, desde que a base de dados de destino esteja *vazia*.
2. */include*
   A pasta `include` armazena todos os arquivos que serão executados na base de dados, independentemente de sua configuração. Qualquer script presente nesta pasta será executado se ainda não estiver registrado na tabela de controle de migration da base de dados alvo.
3. */test*
   Os arquivos na pasta `test` são executados na base de dados alvo apenas quando a flag de teste está em uso.

## Configuração do Migration

Os arquivos `.sql`, compatíveis com SQL Plus, devem ser adicionados às pastas após a realização de testes e checagem de sintaxe, garantindo sua execução correta no contexto de migration.

### Rastrear Issues

Para registro e rastreamento de issues, os arquivos `.sql` adicionados ao repositório podem incluir um comentário que atua como metadado. Este será utilizado pelo sistema de migration para registrar a informação na tabela de controle do banco de dados.

A estrutura do metadado é a seguinte: `-- Issue: link-para-issue`

O metadado é o link da issue, que deve ser registrado na primeira linha do arquivo sql conforme exemplo abaixo para o arquivo `/setup/20240001-1-Status.sql`:

sql

```
-- Issue: api-docsflow#282

-- Status
CREATE TABLE STATUS (
    ID NUMBER(3, 0) NOT NULL,
    NOME VARCHAR2(20) NOT NULL,
    DESCRICAO VARCHAR2(30) NULL,
    PRIMARY KEY (ID),
    UNIQUE (NOME)
);
```

### Filtragem de Branches

Em determinados cenários, pode ser preciso criar scripts que são relevantes apenas para execução em ambientes específicos, não em todos. O pipeline de migrations atende a essa necessidade por meio do metadado `Only`, que deve ser inserido no arquivo SQL.

O formato do metadado é: `-- Only: <ambientes>`, onde `<ambientes>` é uma lista de ambientes separados por vírgulas. Os ambientes possíveis são: `DEV`, `HML` e `PRD`.

No exemplo mencionado, se for necessário executar o script somente no ambiente de produção, você pode adicionar o metadado `Only` para facilitar essa tarefa.

sql

```
-- Issue: api-docsflow#282
-- Only: PRD

-- Status
CREATE TABLE STATUS (
    ID NUMBER(3, 0) NOT NULL,
    NOME VARCHAR2(20) NOT NULL,
    DESCRICAO VARCHAR2(30) NULL,
    PRIMARY KEY (ID),
    UNIQUE (NOME)
);
```

Você pode registrar múltiplos ambientes para execução, incluindo cada um deles separadamente com uma vírgula. No exemplo abaixo, a execução em ambiente de DSV será ignorada.

sql

```
-- Issue: api-docsflow#282
-- Only: HML, PRD

-- Status
CREATE TABLE STATUS (
    ID NUMBER(3, 0) NOT NULL,
    NOME VARCHAR2(20) NOT NULL,
    DESCRICAO VARCHAR2(30) NULL,
    PRIMARY KEY (ID),
    UNIQUE (NOME)
);
```

## Delimitador de Comandos SQL

É obrigatório a utilização de um delimitador quando há múltiplos comandos em um mesmo arquivo SQL:

No exemplo abaixo, as colunas são criadas na tabela status:

sql

```
-- Issue: api-docsflow#282

-- Status
ALTER TABLE STATUS ADD CATEGORIA_ID NUMBER(3, 0);
ALTER TABLE STATUS ADD CHAVE VARCHAR2(30);
```

### Script Sem Delimitador de Comandos

Script SQL sem delimitador de comandos:

sql

```
-- Issue: api-docsflow#282

-- Status
ALTER TABLE STATUS ADD CATEGORIA_ID NUMBER(3, 0)
ALTER TABLE STATUS ADD CHAVE VARCHAR2(30)
```

No exemplo mencionado, as colunas não são criadas devido à ausência de um delimitador. O SQL Plus não retorna um status de erro para aquarda a conclusão do comando com um delimitador. Devido a isso, ocorre um problema conhecido: o registro é criado na tabela de controle com o status `3 - Executado com sucesso`, mesmo que a operação não tenha sido bem-sucedida.

### Bloco PL/SQL

A barra (/) é usada para executar uma instrução SQL ou bloco PL/SQL que está atualmente no buffer. Deve ser utilizada na criação/alteração de Trigger, Procedure ou bloco PL/SQL delimitado por BEGIN - END:

sql

```
-- Issue: api-docsflow#282

-- Status
BEGIN
    INSERT INTO STATUS VALUES (1, 'Ativo', 'Status Ativo');
    INSERT INTO STATUS VALUES (1, 'Inativo', 'Status Inativo');
END;
/
```

- *No exemplo acima, as linhas são inseridas na tabela STATUS em uma única transação*
- ***Atenção: Caso a barra não seja colocada devidamente em uma "nova linha" como consta acima, os comandos foram apenas no buffer e não executados pelo SQL+Plus***
- *Nesse caso o log do SQL+Plus exibe a mensagem "**Session altered**" indicando apenas que os comandos foram colocados no buffer*

```
Connected to:
Oracle Database 19c EI Extreme Perf Release 19.0.0.0.0 - Production
Version 19.14.0.0.0
SQL> SQL>
Session altered.
SQL> SQL> 2#  Disconnected from Oracle Database 19c EI Extreme Perf Release
19.0.0.0.0 - Production
```

## Tabela de Controle

Durante o processo de criação da base, ou no primeiro registro do migration, o processo **verifica se existe na base de dados** uma tabela de controle, cujo nome padrão é `__BASE_MIGRATION`.

A tabela abaixo exemplifica a estrutura da tabela de controle:

| PROJECTNAME | THEME | FILENAME | FILEDATE | REGISTERDATE | ISSUE |
|---|---|---|---|---|---|
| api-docsflow | setup | 20230901-1-Base.sql | 20230901 | 2023-09-01 09:25:51 | api-docsflow#202 |
| api-docsflow | setup | 20230901-2-Status.sql | 20230901 | 2023-09-01 09:25:51 | api-docsflow#202 |

Nesta tabela, é possível consultar informações do arquivo, o script registrado para execução (sm base64), metadados da issue, sistema e datas de execução.

## Fila de Processamento

A ativação do processamento de migration ocorre por meio de comandos de pipeline. No entanto, um script pode ocasionalmente falhar, resultando em um job mal-sucedido. Para solucionar este problema e liberar a fila, sugere-se o seguinte procedimento:

1. Atualize diretamente na branch o arquivo que apresentou falha, substituindo-o por uma versão corrigida.
2. Execute o pipeline no GitLab, passando o parâmetro `MIGRATIONS_FIX = 1`.
3. Execute novamente no GitLab o job que falhou.

Dessa maneira, a fila de processamento será liberada e os scripts subsequentes serão executados em seus respectivos merge requests.

### Status de Execução

O indicador de execução do script também é apresentado no campo IDSTATUS:

| Status | Significado |
|---|---|
| 0 | Não Executado |
| 1 | Executado com sucesso |
| 3 | Executado com erro |

Next page
Ambiente

On this page
Nomenclatura de Projetos de...
Organização de Grupos de P...
Documentação Geral e Wiki
Gestão de Branches
Configuração de Pipeline
Estrutura
Configuração do Migration
Delimitador de Comandos SQL
Tabela de Controle
Fila de Processamento

Made with Material for MkDocs

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's
Design de contrato de API's
Monitoramente e

# Ambiente Openshift

**On this page**
Introdução
Acesso ao OCP
Arquitetura do OCP
Gerenciamento do Ambiente
Recursos Adicionais

## Introdução

O Banco da Amazônia adota o Openshift para a implantação e gerenciamento de novas Aplicações. O OpenShift é uma plataforma de computação em nuvem baseada em contêineres, desenvolvida pela Red Hat. Ela fornece um ambiente de desenvolvimento e implantação de aplicativos altamente escalável e flexível, utilizando tecnologias como o Kubernetes, Docker e outras ferramentas de código aberto.

O principal objetivo do OpenShift é simplificar e automatizar o processo de desenvolvimento, implantação e gerenciamento de aplicativos em contêineres. Ele oferece recursos avançados para orquestração de contêineres, permitindo que os desenvolvedores construam, testem e implantem suas aplicações de forma mais eficiente.

No geral, o OpenShift simplifica o processo de implantação e gerenciamento de aplicativos em contêineres, fornecendo uma plataforma unificada que ajuda os desenvolvedores a acelerar o ciclo de desenvolvimento, reduzir a complexidade operacional e obter maior escalabilidade e flexibilidade em seus aplicativos.

O OpenShift Container Platform (OCP) é a versão corporativa do OpenShift, desenvolvida pela Red Hat. O OCP oferece todas as funcionalidades do OpenShift, juntamente com recursos adicionais que atendem às necessidades específicas de organizações de grande porte

## Acesso ao OCP

Atualmente existem dois ambientes de Cluster Openshift. Um que utiliza a versão 3, e outro que utiliza a versão 4.

### Acesso via Web Console

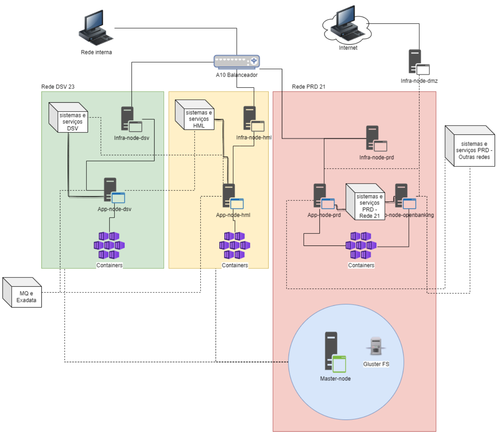
Segue os links para acesso a cada ambiente:

- OpenShift 3;
  - Acesso
- OpenShift 4;
  - Homologação / Desenvolvimento: Acesso
  - Produção: Acesso

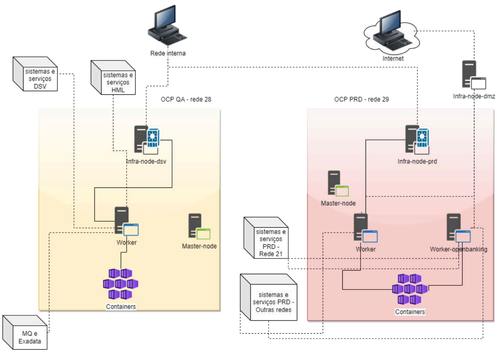
## Arquitetura do OCP

A seguir seguem os diagramas de arquitetura dos ambientes OCP 3 e OCP 4:

### OCP 3




### OCP 4




## Gerenciamento do Ambiente

Os ambientes OCP3 e OCP4 do Banco da Amazônia são gerenciados pela GPROD-COBAN. Então, para esta coordenadoria, deve-se abrir chamados para as seguintes solicitações:

- Acesso aos projetos no OCP 3 e OCP 4;
- Criação de Infraestrutura (Novos projetos);
- Exclusão de projetos;
- E outros relacionados ao OCP.

## Recursos Adicionais

### Links Úteis

- Openshift 4 Documentação
- Getting Started CLI;

Created by JUST ONE PAGE PDF

Banco de Dados
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

OpenShift
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

MQ
Concentos e Ambiente

Ferramentas
Catálogo de API's
Simulação e teste de API's
Design de contrato de API's
Monitoramente e

# Criação de Ambientes de Deployment no OpenShift

On this page
Solicitação de Infraestrutura ...
Repositório do Projeto no Git...
Documentações Plano de Ar...
Abertura de Chamado

## Solicitação de Infraestrutura para Projetos

Para a solicitação de Infraestrutura de novos projetos é necessário seguir os seguintes passos:

1. Criar o repositório do Projeto no Gitlab;
   - Responsável: Área de Sistemas;
   - Detalhes: Repositório do Projeto no GitLab.
2. Criar Plano de Arquitetura (PLA) e Plano de Infraestrutura (PLI);
   - Responsável: Área de Sistemas;
   - Detalhes: Documentações PLA e PLI.
3. Abrir chamado para a GEGOV/COARQ solicitando a Aprovação do PLA;
   - Responsável: Área de Sistemas;
   - Detalhes: Chamado para o PLA.
4. Providenciar a Aprovação do PLA;
   - Responsável: GEGOV/COARQ (GPROT/COAGD).
5. Abrir chamado para a GPROD/COBAN solicitando a Aprovação do PLI.
   - Responsável: Área de Sistemas.
   - Detalhes: Chamado para o PLI.
6. Providenciar a Aprovação do PLI;
   - Responsável: GPROD/COBAN;
   - Requisito: PLA Aprovado.

## Repositório do Projeto no GitLab

O GitLab é a plataforma de gerenciamento do ciclo de vida de desenvolvimento de software (SDLC) utilizada pelo Banco da Amazônia. Ele fornece um conjunto abrangente de recursos para auxiliar equipes de desenvolvimento no versionamento de código, colaboração, CI/CD (Integração Continua/Entrega Continua), rastreamento de problemas, gerenciamento de projetos e etc.

Para que um novo projeto seja criado na Infraestrutura Openshift 4 do Banco da Amazônia, é mandatório a existência do projeto versionado no GitLab. O Basa possui duas versões do Gitlab em produção, porém, para novos projetos, só a versão mais atualizada poderá ser utilizada, a versão que se encontra no link abaixo:

- GitLab

O nome do projeto no GitLab deverá obedecer o padrão de nomenclatura que está detalhado no link abaixo:

- Nomenclatura de Projetos Gitlab

## Documentações Plano de Arquitetura (PLA) e Plano de Infraestrutura (PLI)

Os documentos Plano de Arquitetura e Plano de Infraestrutura são obrigatórios para a solicitação de criação dos ambientes de deployment e devem ser criados na Wiki de projeto no GitLab. Abaixo segue os templates para cada documento:

- Plano de Arquitetura
- Plano de Infraestrutura

O PLA deve ser o primeiro documento a ser aprovado, através de abertura de chamado para a GEGOV/COARQ. Em seguida, deve-se abrir chamado para a GPROD/COBAN para aprovação do PLI e criação do Ambiente de deplomento.

As seguintes informações são obrigatórias:

- No Plano de Arquitetura (PLA);
  - Imagem;
    - Ex: nginx-114-rhel7;
    - Detalhes: Opções de Image Stream.
  - Linguagem de Programação.
    - Ex: Javascript ECMAScript.
- No Plano de Infraestrutura (PLI).
  - Nome do projeto de acordo com a nomenclatura padrão;
  - Necessidade de criação de usuário/schema no Banco de Dados.

### Opções de Image Stream

Como dito anteriormente, a informação da imagem do projeto é obrigatória no Plano de Arquitetura (PLA). Segue lista das imagens disponiveis para a criação de infraestrutura:

| Tecnologia | Nome do Image Stream |
|---|---|
| Java: | redhat-openjdk18-openshift:1.5 |
| | openjdk-11-rhel7:1.10 |
| | openjdk-17:1.11-2 |
| Vuejs: | nginx-120-rhel8:1-36 |
| Jboss: | basa-custom-eap71:1.0 |
| | basa-custom-eap71:recalculo |
| | basa-custom-eap72:1.0 |
| | basa-custom-eap73:1.0 |
| Dotnet: | dotnet:3.1-20 |

## Abertura de Chamado

### Aprovação de PLA

O chamado deverá ser categorizado com as seguintes opções:

- Portal: Serviços de TI;
- Categoria: Sistemas;
- Subcat.: Sistemas bancários;
- Detalhe: Validar plano de arquitetura;
- Detalhe II: API;
- Ambiente: Produção.

O chamado precisa conter obrigatoriamente as seguintes informações:

- Link do PLA para aprovação

A seguir um exemplo de descrição para o chamado:

```md
PLANO DE ARQUITETURA - CADOC (FRONTEND) E API-CADOC BACKEND)

Prezados,

Favor providenciar verificação e aprovação do PLA para o sistema CADOC (FRONTEND) E API-CADOC BACKEND)
respectivamente:

https://gitlab.bancoamazonia.sa/sistemas/credito/spdd/cadoc/-/wikis/Plano-de-Arquitetura
https://gitlab.bancoamazonia.sa/sistemas/credito/spdd/api-cadoc/-/wikis/Plano-de-Arquitetura
```

O chamado deverá ser encaminhado para GEGOV/COARQ (GPROT/COAGD).

### Aprovação de PLI

O chamado deverá ser categorizado com as seguintes opções:

- Portal: Serviços de TI;
- Categoria: Sistemas;
- Subcat.: Sistemas bancários;
- Detalhe: Validar plano de arquitetura;
- Detalhe II: API;
- Ambiente: Produção.

O chamado precisa conter obrigatoriamente as seguintes informações:

- Link do PLI para aprovação

A seguir um exemplo de descrição para o chamado:

```md
PLANO DE INFRAESTRUTURA - API-CADOC
Prezados,

Favor providenciar a validação e aprovação do plano de insfraestrutura para o sistema CADOC:

API:
https://gitlab.bancoamazonia.sa/sistemas/credito/spdd/api-cadoc/-/wikis/Plano-de-Infraestrutura

FRONT:
https://gitlab.bancoamazonia.sa/sistemas/credito/spdd/cadoc/-/wikis/Plano-de-Infraestrutura
```

O chamado deverá ser encaminhado para GPROD/COBAN.



Powered by MkDocs and Material for MkDocs
Created by MKDOCS ADMIN

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's

# Solicitação de Acesso

On this page
Abertura de chamado

## Abertura de chamado

O chamado deverá ser categorizado com as seguintes opções:

- Portal: Serviços de TI;
- Categoria: Servidores;
- Subcat.: Servidores de aplicação;
- Detalhe: Manutenção da aplicação.

O chamado precisa conter obrigatoriamente as seguintes informações:

- Nome do projeto e/ou link na console;
- Matrículas que precisam do acesso;
- Nome da empresa (No caso de usuários terceirizados).

Next page
Ambiente

Mon Nov 25 2024 17:36:24 GMT-0300 (Brasilia Standard Time)
Created by JUST ONE PAGE PDF

Manual de Desenvolvimento

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's



# Erros Comuns

On this page
Falha no Estágio Initialized d...
Reexecução do Pipeline no G...

## Falha no Estágio Initialized do Pipeline

Alguns problemas podem ocorrer no estágio Initialized da execução do pipeline de uma aplicação. Muitos deles são causados pela configuração do gitlab-ci da branch em questão. Dependendo do tipo de projeto ou biblioteca utilizada, o arquivo gitlab-ci deve ser configurado de forma específica. Então, nesses casos, quando encontramos problemas nesse estágio recomenda-se primeiramente:

1. Verificar se o arquivo gitlab-ci existe na raiz da branch no qual o pipeline está sendo executado;
2. Verificar se o arquivo gitlab-ci está preenchido de acordo com o Manual de Desenvolvimento de Software.
   - GitLab CI para projetos Java;
   - GitLab CI para projetos VueJs.

## Reexecução do Pipeline no Gitlab utilizando variáveis no cache

Alguns problemas podem permanecer no pipeline mesmo após serem resolvidos. Isso acontece porque durante a reexecução de um pipeline o gitlab mantém algumas informações desatualizadas. Por exemplo, quando executamos um pipeline sem determinada configuração (token, gitlab-ci...) e após fazer a configuração correta e reexecutar o pipeline existente, o erro continua. Então nesses casos, recomendamos executar o pipeline do zero utilizando os seguintes passos:

1. Na página do projeto clicar em CI/CI > Pipelines;
2. Na página dos pipelines clicar no botão "Run Pipeline";
3. Na página do "Run Pipeline" escolher a branch e clicar no botão "Run Pipeline".

Next page
Ambiente

Mon Nov 25 2024 17:28:58 GMT-0300 (Brasília Standard Time)
Created by JUST ONE PAGE PDF

https://manual-desenvolvimento-software.apps.ocp-prd.bancoamazonia.sa



**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's

# Conceitos no IBM MQ

O IBM MQ é um sistema de mensageria que utiliza diversos conceitos importantes para garantir a comunicação confiável e segura entre aplicativos distribuídos. Neste artigo, exploraremos três desses conceitos fundamentais: gerenciadores de filas, filas e canais.

On this page
Gerenciadores de Filas
Filas
Canais
Ambiente MQ Series 8.0
Ambiente MQ Series 5.3

## Gerenciadores de Filas

Um gerenciador de filas é o componente central do IBM MQ. Ele é responsável por controlar o armazenamento e a entrega de mensagens entre os aplicativos. O gerenciador de filas mantém filas locais, onde as mensagens são temporariamente armazenadas até serem processadas pelos aplicativos de destino. Ele também gerencia filas remotas e de transmissão, que veremos a seguir.

## Filas

As filas são estruturas de armazenamento usadas para o intercâmbio de mensagens no IBM MQ. Existem três tipos principais de filas:

1. **Filas Locais**: São filas dentro do gerenciador de filas, onde as mensagens são enfileiradas e aguardam para serem processadas pelos aplicativos de destino. As mensagens são colocadas nessa fila por produtores e consumidas pelos consumidores.
2. **Filas Remotas**: São filas localizadas em outros gerenciadores de filas, que podem estar em diferentes sistemas ou locais físicos. Elas permitem a comunicação entre aplicativos em diferentes ambientes. As mensagens enviadas para filas remotas são transmitidas através de canais de conexão.
3. **Filas de Transmissão**: Também conhecidas como *transmission queues*, são usadas para enviar mensagens de um gerenciador de filas para outro. Elas permitem o encaminhamento de mensagens de forma confiável, garantindo a entrega em caso de falhas temporárias ou interrupções de rede.

## Canais

Os canais são mecanismos de comunicação utilizados pelo IBM MQ para transmitir mensagens entre gerenciadores de filas. Existem três tipos principais de canais:

1. **Canais de Conexão**: Também conhecidos como *channel connections*, são canais bidirecionais que estabelecem a comunicação entre dois gerenciadores de filas. Eles permitem a troca de mensagens entre filas remotas.
2. **Canais Emissores**: Também conhecidos como *sender channels*, são canais unidirecionais utilizados para enviar mensagens de um gerenciador de filas para outro. Eles são responsáveis pela transmissão das mensagens de uma fila local para uma fila remota.
3. **Canais Receptores**: Também conhecidos como *receiver channels*, são canais unidirecionais que recebem mensagens de um gerenciador de filas remoto e as entregam a uma fila local. Eles são responsáveis por receber as mensagens transmitidas pelos canais emissores.

Os canais garantem a entrega confiável e segura das mensagens entre os gerenciadores de filas, facilitando a comunicação entre aplicativos distribuídos.

## Ambiente MQ Series 8.0

O ambiente MQ Series 8.0 está hospedado no servidor srv5520 - IP 10.5.5.20 e possui os seguintes gerenciadores:

- QM.22222222.01 (porta 1415) - Automação (banklink)
- QM.AMZ (desativado) - Amazoncred
- QM.AUTLAN (porta 1439) - Automação (banklink)
- QM.AUTBAN (porta 1435) - Automação (banklink)
- QM.RGT (porta 1438) - Automação (banklink) Registrato
- QM.ACW (porta 1442) - ACW
- QM.AMZ.02 (porta 1443) - Amazoncred

## Ambiente MQ Series 5.3

O ambiente MQ Series 5.3 está hospedado no servidor srvmqprd - IP 10.5.5.15 e possui os seguintes gerenciadores:

- QM.04902979.01 (porta 1414) - SPB
- QM.04902979.02 (porta 12426) - MES01
- QM.04902979.03 (porta 13422) - DDA/Cheque Legal (MES02)
- QM.33333333.01 (porta 1417) - Compulsório
- QM.55555555.01 (porta 1420) - CCOR
- QM.66666666.01 (porta 1424) - COMPE
- QM.CARDWARE (porta 1431) - CARDWARE
- QM.FUNDOSNO (porta 1425) - FUNDOS
- QM.IBANKING (porta 1422) - IBK
- QM.INTOLLER (porta 1423) - INTOLLERANT
- QM.NETBANKG (porta 1421) - AMAZONCRED
- QM.RECOR (porta 1428) - RECOR
- QM.77777777.01 (porta 1426) - AMAZONCAD
- QM.LCA (porta 1437) - LCA
- QM.SDB (porta 1436) - SDB
- QM.CONTROPER (porta 1450) - CONTROPER
- QM.DDABNF.02 (porta 12427) - DDABNF
- QM.SCG (porta 1440) - SCG (Garantias) - Sistema: recebíveis (srv2127 e srv2128)
- QM.04902979.05 (porta 15422) - NPC

Next page
Ambiente

Created by JUST ONE PAGE .PDF

Manual de Desenvolvimento

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's
Design de contrato de API's
Monitoramente e Observabilidade



# Catálogo de API's

O Apicurio Registry é uma solução open-source desenvolvida pela Red Hat para o gerenciamento de esquemas e contratos de APIs. Ele atua como um repositório centralizado onde os desenvolvedores podem armazenar, versionar e gerenciar artefatos relacionados a APIs, como esquemas de dados, contratos de APIs (especificações OpenAPI), definições de eventos (por exemplo, em Apache Kafka), e outros tipos de metadados. Essa aplicação foi implantada para funcionar como catálogo de API's no contexto dos negócios do Banco da Amazônia para agilizar o processo de gestão de API's. É possível acessar o documento de habilitação das equipes neste link e também a documentação oficial da ferramenta aqui.

Para acessar a aplicação, siga este link.

Next page
Ambiente

Mon Nov 25 2024 17:28:36 GMT-0300 (Brasília Standard Time)
Created by JUST ONE PAGE PDF

Manual de Desenvolvimento

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's



# Simulação e Teste de API's



Microcks é uma ferramenta open-source usada para testar APIs e sistemas baseados em eventos, além de simular (mock) seus comportamentos. O seu foco é a simulação e teste de microsserviços, APIs REST e SOAP, bem como contratos de eventos em plataformas como Apache Kafka. Essa ferramenta poderá contribuir para a construção de mocks corporativos futuramente. Mas, ela também é muito útil no processo inicial de definições de uma solução para simular APIs que ainda não estão disponíveis quanto para testar automaticamente a conformidade e desempenho de APIs e mensagens em arquiteturas modernas baseadas em microserviços e eventos

Para acessar a aplicação, siga este link.

Next page
Ambiente

Mon Nov 25 2024 17:28:59 GMT-0300 (Brasilia Standard Time)
Created by MUST ONE PAGE PDF

Manual de Desenvolvimento

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's
Design de contrato de API's



# Design de contrato de API's

O API Designer é útil para modelar APIs de forma colaborativa e visual, facilitando a criação de especificações de APIs (como OpenAPI ou Swagger). A principal finalidade dessa ferramenta é permitir que as equipes, arquitetos, analistas e outros stakeholders definam o contrato de uma API antes que sua implementação comece, garantindo clareza e padronização no design.

Para acessar a aplicação, siga este link.

Next page
Ambiente

Mon Nov 25 2024 17:26:42 GMT+0300 (Brasília Standard Time)
Created by JUST ONE PAGE PDF

Manual de Desenvolvimento

**Banco de Dados**
Ambiente
Solicitação de Acesso
Erros Comuns
Database Migration

**OpenShift**
Ambiente
Solicitação de Infraestrutura
Solicitação de Acesso a Projetos
Erros Comuns

**MQ**
Concentos e Ambiente

**Ferramentas**
Catálogo de API's
Simulação e teste de API's



# Monitoramente e observabilidade

O Grafana a ferramenta open-source que o BASA implantou para a visualização de dados e monitoramento de sistemas. Ela é amplamente usada em TI, DevOps e engenharia de software. E permite criar dashboards interativos que exibem métricas de desempenho em tempo real, como uso de CPU, memória, taxa de erros e tempo de resposta de serviços. Integrando-se a diversas fontes de dados como Prometheus, InfluxDB e Elasticsearch, o Grafana oferece suporte para monitoramento de infraestrutura, aplicações, logs e sistemas baseados em eventos. Com alertas configuráveis, ele ajuda a detectar problemas de performance antecipadamente. Muito utilizado em arquiteturas de microserviços e monitoramento de infraestrutura, ele é uma ferramenta essencial para garantir observabilidade e resolução proativa de problemas.

Para acessar a aplicação, siga este link.

Next page
Ambiente

Mon Nov 25 2024 17:36:46 GMT-0300 (Brasilia Standard Time)
Created by JUST ONE PAGE PDF