Swarm Cluster (Overview / Welcome)

Visão Geral do Cluster

O Swarm Cluster é a plataforma de execução de aplicações, APIs, serviços internos e ferramentas operacionais da empresa.

Ele foi construído utilizando containers Docker e orquestração Docker Swarm, com o objetivo de fornecer um ambiente padronizado, previsível e observável para execução de software.

A plataforma foi projetada seguindo alguns princípios fundamentais:

- Infraestrutura como código
- Deploy versionado
- Observabilidade centralizada
- Isolamento entre aplicações
- Segurança por camadas
- Simplicidade operacional
- Reprodutibilidade de ambientes
- Escalabilidade controlada
- Baixo acoplamento entre serviços

O cluster funciona como uma plataforma interna onde novas aplicações podem ser publicadas de forma padronizada, utilizando imagens Docker versionadas e deploy via stacks.

A infraestrutura também fornece serviços compartilhados como:

- Proxy reverso
- TLS automático
- Logs centralizados
- Métricas e dashboards
- Autenticação SSO
- Storage comparilhado
- Cache Redis
- Web Application Firewall

Dessa forma, os desenvolvedores podem focar nas aplicações enquanto o cluster fornece toda a base operacional necessária.

Engine do Cluster

Engine Operacional do Cluster

O cluster opera baseado em alguns pilares fundamentais:

Orquestração

Responsável por manter containers rodando, reiniciar serviços, distribuir cargas e gerenciar deploys.

Tecnologia: Docker Swarm

Rede

Rede overlay interna permitindo comunicação entre serviços independentemente do nó.

Storage

Volumes persistentes via AWS EFS compartilhado entre todos os nós.

Observabilidade

Logs, métricas e alertas centralizados.

Segurança

TLS automático, WAF, autenticação SSO e isolamento por redes internas.

Deploy

Deploy versionado via imagens Docker e stacks.

Arquitetura

Componente	**Função**
Traefik	Proxy reverso / entrada HTTP
Docker Swarm	Orquestração de containers
Portainer	Gerenciamento visual
Loki	Armazenamento de logs
Alloy	Coleta de logs

Componente	Função
Prometheus	Métricas
Grafana	Dashboards
EFS	Storage compartilhado
Redis	Cache e sessões
Authentik	SSO
WAF	Web Application Firewall

Fluxo de requisição

```
Internet                    Internet
   ↓                           ↓
Traefik                     Traefik
   ↓                           ↓
WAF                         WAF
   ↓                           ↓
NGINX                       GO APP
   ↓
PHP-FPM/APP
```

Stack de Infraestrutura

Introdução

A Stack de Infraestrutura reúne o conjunto de serviços compartilhados disponibilizados pelo cluster para atender necessidades comuns das aplicações e das operações internas.

Embora cada projeto possa, conforme seus requisitos, publicar componentes próprios de infraestrutura, como bancos de dados, serviços de cache ou ferramentas auxiliares, o cluster já oferece um pacote inicial de recursos reutilizáveis. Esse modelo reduz o esforço de implantação, evita duplicação desnecessária de serviços e promove maior padronização operacional entre as stacks.

Esses serviços não substituem totalmente a possibilidade de infraestruturas dedicadas por aplicação. Em cenários que exijam isolamento, requisitos específicos de desempenho, compatibilidade de versão, compliance ou autonomia operacional, cada stack poderá prover seus próprios componentes. Ainda assim, para boa parte dos cados, o conjunto compartilhado do cluster atende de forma satisfatória as demandas iniciais e intermediárias dos projetos.

A seguir, são apresentados os principais serviços de infraestrutura atualmente disponibilizados.

Stack de Infraestrutura

Proxy de Entrada e Roteamento

O acesso HTTP e HTTPs ao cluster é centralizado pelo **Traefik**, que atua como proxy reverso de borda. Ele é responsável por receber as requisições externas, aplicar o roteamento com base em domínio e encaminhar o tráfego para os serviços corretos dentro do ambiente.

Além do roteamento, o Traefik também concentra responsabilidades como terminação TLS, publicação padronizada de aplicações web e integração com regras de exposição controlada.

Esse componente representa a porta de entrada principal do cluster para aplicações publicadas via web.

Gerenciamento Operacional

O **Portainer** é disponibilizado como interface de gerenciamento visual do ambiente Docker Swarm. Sua função é facilitar inspeções operacionais, acompanhamento de serviços, análise de containers, volumes, redes e demais recursos do cluster.

Embora o gerenciamento automatizado e o deploy padronizado devam privilegiar CLI e versionamento em arquivos declarativos, a presença do Portainer reduz a dependência exclusiva de acesso terminal para inspeções e ações operacionais controladas.

Storage Compartilhado

O cluster disponibiliza armazenamento compartilhado em rede para aplicações que necessitem persistência de arquivos entre réplicas ou entre diferentes serviços.

Esse armazenamento é provido por volumes montados sobre **NFS/EFS**, permitindo uso compartilhado em cenários como :

- upload de arquivos
- armazenamento persistente de dados de aplicações
- diretórios compartilhados entre serviços
- persistência de artefatos operacionais
- volumes acessíveis por múltiplos nós

A existência dessa camada compartilhada simplifica a execução de aplicações stateful que precisem manter arquivos fora do ciclo de vida efêmero dos containers.

Navegação de arquivos em volumes compartilhados

Para facilitar a inspeção e navegação de arquivos persistidos nos volumes montados em NFS, o cluster também disponibiliza um frontend web baseado em **FileBrowser**.

Esse serviço permite visualizar, enviar, baixar e organizar arquivos armazenados nos diretórios compartilhados, o que pode ser útil para apoio operacional, conferência de uploads, validações manuais e administração de conteúdos persistidos por aplicações.

Bancos de Dados Compartilhados

O cluster já oferece serviços de banco de dados que podem ser utilizados por aplicações quando não houver necessidade de instância dedicada.

Atualmente, estão disponíveis:

- MariaDB
- PostgreSQL

Esses serviços compartilhados podem ser úteis para aplicações internas, ferramentas administrativas, provas de conceito, ambientes de desenvolvimento ou sistemas cujo perfil não exija isolamento completo da camada de banco.

Ainda assim, cada projeto deve avaliar cuidadosamente se o uso compartilhado é adequado ao seu contexto. Aplicações com exigências específicas de performance, segurança, versionamento ou governança podem demandar bancos dedicados.

Ferramenta de Acesso a Banco de Dados

Para facilitar a administração dos bancos disponibilizados no cluster, também existe o **CloudBeaver**, que fornece uma interface web para navegação, consulta e administração de diferentes motores de banco de dados.

Seu objetivo é oferecer um ponto centralizado e amigável para operações de inspeção, troubleshooting e apoio ao desenvolvimento, reduzindo a necessidade de acesso direto por clientes externos em cenários simples.

Cache e Dados Temporários

O cluster disponibiliza **Redis** como serviço compartilhado para uso por aplicações que necessitem mecanismos de cache, armazenamento temporário, filas leves, controle distribuído ou compartilhamento de sessão.

Esse recurso é especialmente útil em aplicações web com múltiplas réplicas, onde sessões e cache precisam ser centralizados para manter consistência entre instâncias.

Assim como nos demais componentes, nada impede que uma aplicação use uma instância própria de Redis quando houver necessidade específica de isolamento ou configuração dedicada.

Autenticação e Federação de Identidade

O cluster disponibiliza o **Authentik** como componente central para autenticação e federação de identidade.

Seu papel é atuar como broker entre aplicações internas e diferentes provedores de autenticação, permitindo integrar o ambiente ao sistema de identidade da empresa e a fluxos de autenticação federada.

Essa abordagem favorece a padronização de acesso aos serviços internos e reduz o acoplamento direto de cada aplicação com provedores externos específicos. Em vez de cada sistema implementar integrações independentes, o Authentik passa a intermediar essa relação, centralizando autenticação, políticas e governança de acesso.

Em termos práticos, isso permite:

- centralizar autenticação de aplicações internas
- federar o SSO corporativo
- desacoplar aplicações dos provedores externos
- facilitar desativação de acesso de usuários
- preparar o ambiente para evolução gradual rumo a SSO mais consistente

Gestão de Segredos e Dados Sensíveis

O cluster também prevê o uso de ferramentas voltadas à gestão de dados sensíveis, entre elas o **Vaultwarden**.

Esse componente pode ser utilizado como apoio à administração segura de credenciais, segredos operacionais e informações restritas relacionadas ao ecossistema da plataforma, sempre observando os critérios internos de segurança e governança.

Observabilidade Básica Compartilhada

O cluster oferece uma base padronizada de observabilidade aplicada especialmente aos logs emitidos na saída padrão dos containers.

Essa capacidade permite centralizar e consultar logs de serviços executados na plataforma, fornecendo suporte a troubleshooting, inspeção operacional e análise de falhas.

Como a observabilidade possui arquitetura, objetivos e componentes próprios, sua descrição detalhada é apresentada em seção específica, evitando repetição nesta página.

Papel da Stack de Infraestrutura

A Stack de Infraestrutura deve ser entendida como um conjunto de serviços-base oferecidos pelo cluster para acelerar a publicação e a operação de aplicações.

Seu objetivo não é impor um modelo único para todos os projetos, mas disponibilizar um ponto de partida padronizado e funcional. Cada aplicação pode consumir esses recursos compartilhados ou optar por implantar componentes próprios, de acordo com suas necessidades técnicas e operacionais.

Em resumo, essa stack existe para:

- reduzir esforço inicial de novos projetos
- evitar duplicação desnecessária de serviços
- concentrar ferrametas comuns em um ambiente conhecido
- promover padronização operacional
- simplificar integrações recorrentes
- oferecer uma base de infraestrutura pronta para uso

Engine da Observabilidade

Introdução

A Engine de Observabilidade do cluster é o conjunto de serviços responsáveis por coleta, armazenamento, consulta e visualização de logs, métricas e eventos operacionais de toda a plataforma.

O objetivo da observabilidade é permitir:

- Diagnóstico rápido de problemas
- Monitoramento de serviços
- Análise de comportamento de aplicações
- Auditoria de eventos
- Criação de alertas
- Visualização de méttricas
- Troubleshooting operacional
- Análise histórica de falhas e incidentes

A observabilidade é tratada como parte fundamental da infraestrutura, e não como um recurso opcional.

Arquitetura da Observabilidade

A arquitetura da observabilidade do cluster é composta pelos seguintes componentes:

Componente	Função
Alloy	Coleta e roteamento de logs
Loki	Armazenamento e consulta de logs
Prometheus	Coleta de métricas
Grafana	Visualização e dashboards
Node Exporter	Métricas dos servidores
cAdvisor	Métricas dos containers

Fluxo de Logs

`Container → Docker Logging → Alloy → Loki → Grafana`

Fluxo de Métricas

`Node Exporter / cAdvisor → Prometheus → Grafana`

Coleta de Logs com Alloy

Todos os containers do cluster enviam seus logs através do Docker Logging Driver utilizando o protocolo Syslog.

Os logs não são enviados diretamente para o Loki. Existe uma camada intermediária chamada **Alloy**, que atua como agente de coleta e roteamento de logs.

Vantagens dessa arquitetura

- Desacoplamento entre aplicações e backend de logs
- Padronização de logs
- Controle centralizado
- Possibilidade de mudar o backend sem alterar aplicações
- Coletor local em cada nó
- Baixa latência
- Menor tráfego de rede overlay

Cada nó do cluster executa uma instância do Alloy em modo global, funcionando como endpoint local de ingestão de logs.

Os containers enviam logs para:

`tcp://127.0.0.1:51893`

Isso garante que os logs sempre sejam enviados localmente ao nó, evitando dependência de rede overlay.

Armazenamento de Logs com Loki

O Loki é o sistema responsável pelo armazenamento e consulta de logs do cluster.

Diferente de sistemas tradicionais de logs, o Loki não indexa o conteúdo dos logs, apenas labels. Isso reduz drasticamente o consumo e armazenamento.

Os logs são indexados principalmente pelos seguints labels:

- stack
- service
- container
- node
- level (quando disponível)

Isso permite consultas como:

- Logs de um serviço específico
- Logs de uma stack
- Logs de um nó
- Logs de erro
- Logs de um intervalo de tempo

O Loki funciona como backend de logs e é consultado através do Grafana.

Métricas com Prometheus

O Prometheus é responsável pela coleta de métricas do cluster e dos serviços.

As métricas coletadas incluem:

Métricas de Infraestrutura

- CPU
- Memória
- Disco
- Rede
- Load average
- IO

Métricas de Containers

- Uso de CPU por container
- Uso de memória por container
- Network IO
- Restart de containers
- Número de replicas
- Estado de serviços

As métricas são coletadas principalmente através de:

Exporter	**Função**
Node Exporter	Métricas do host
cAdvisor	Métricas dos containers
Prometheus	Coleta e armazenamento

Visualização com Grafana

O Grafana é a interface de visualização da observabilidade.

Atravéd do Grafana é possível:

- Visualizar logs
- Criar dashboards
- Criar alertas
- Monitorar serviços
- Monitorar infraestrutura
- Analisar incidentes
- Visualizar métricas históricas

O grafana funciona como interface única de observabilidade da plataforma.

Filosofia da Observabilidade

A observabilidade do cluster foi projetada seguindo alguns princípios:

Logs são obrigatórios

Todo serviço deve gerar logs.

Logs centralizados

Nenhum log deve ficar apenas dentro do container.

Métricas antes de problemas

Problemas devem ser detectados por métricas antes de usuários perceberem.

Troubleshooting deve ser rápido

Deve ser possível descobrir o problema em minutos, não horas.

Observabilidade faz parte da plataforma

Não é responsabilidade de cada aplicação individual.

Tudo deve ser observável

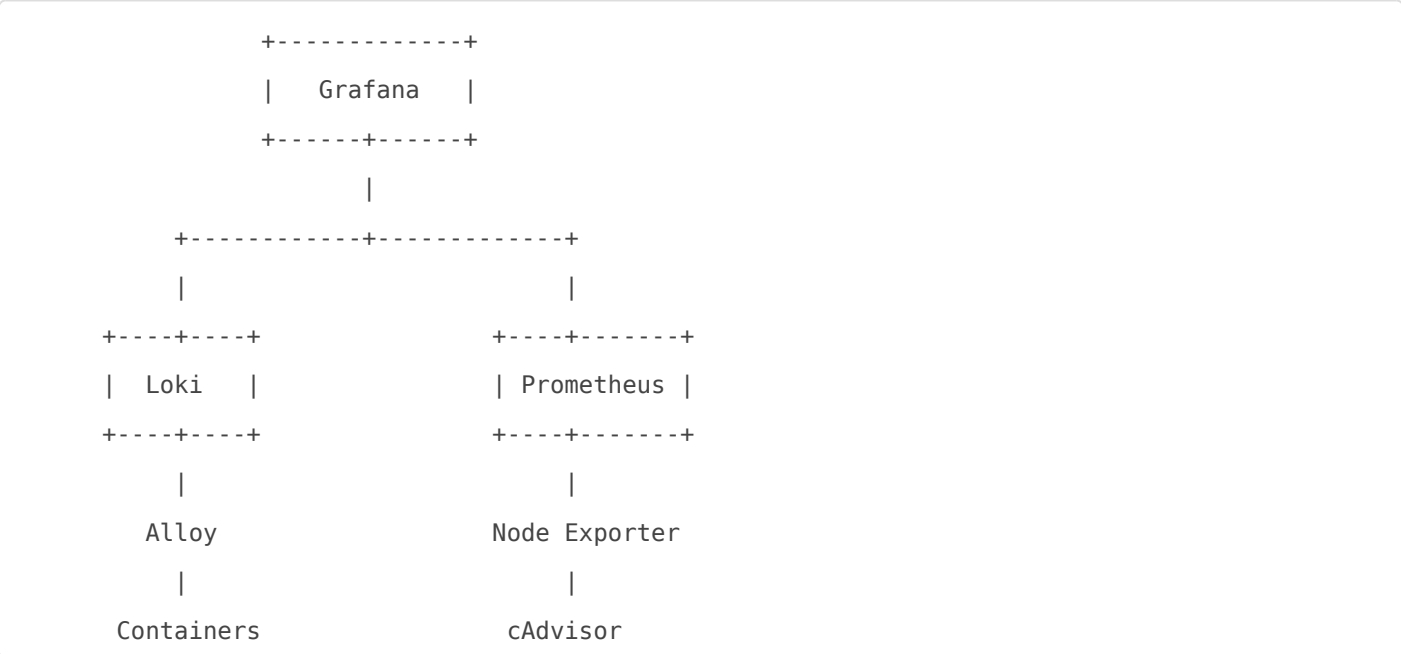
- Infraestrutura
- Containers
- Serviços
- Filas
- APIs
- Banco
- Jobs
- Deploys

Resumo da Engine de Observabilidade

A Engine de Observabilidade do cluster é composta por:

Camada	Ferramenta
Coleta de Logs	Alloy
Armazenamento de Logs	Loki
Coleta de Métricas	Prometheus
Métricas de Host	Node Exporter
Métricas de Containers	cAdvisor
Visualização	Grafana

Diagrama resumido



Por que Docker Swarm e não Kubernetes?

A escolha do Docker Swarm foi baseada em critérios técnicos, operacionais e de complexidade.

Comparação conceitual

Característica	Docker Swarm	Kubernetes
Complexidade	Baixa	Alta
Facilidade de operação	Alta	Média/Baixa
Curva de aprendizado	Baixa	Alta
Overhead de infraestrutura	Baixo	Alto
Ideal para clusters pequenos/médios	Sim	Nem sempre
Tempo de deploy	Muito rápido	Mais complexo
Integração com Docker	Nativa	Indireta
Manutenção	Simples	Complexa

Filosofia adotada

> Escolhemos a ferramenta mais simples que resolve o problema de forma confiável.

O Docker Swarm oferece:

- Orquestração
- Service discovery
- Load balancing interno
- Rolling update
- Secrets
- Configs
- Overlay network
- Deploy declarativo
- Alta disponibilidade

Para o tamanho e necessidades do ambiente, o Swarm atende perfeitamente com muito menos complexidade operacional.

Princípio adotado

> Preferimos um sistema que entendemos completamente do que um sistema extremamente complexo que apenas sabemos operar.

Filosofia de Operação

Introdução

O cluster não é apenas um conjunto de servidores executando containers, mas uma plataforma operacional construída sobre alguns princípios fundamentais.

Esses princípios orientam a forma como aplicações são desenvolvidas, empacotadas, implantadas e operadas dentro do ambiente.

O objetivo dessa filosofia é manter o ambiente previsível, auditável, escalável e operável por qualquer pessoa da equipe com conhecimento da plataforma.

Filosofia de Operação

Infraestutura como Plataforma

O cluster deve ser entendido como uma **plataforma interna de execução de software**, e não apenas como um conjunto de máquinas onde aplicações são instaladas manualmente.

Aplicações não devem depender de configurações manuais em servidores específicos.
Toda aplicação deve ser executada através de containers e publicada via stack.

Isso garante:

- Reprodutibilidade
- Portabilidade
- Versionamento de deploy
- Facilidade de rollback
- Independência de servidor específico
- Escalabilidade controlada

O servidor deixa de ser importante; o que importa é o serviço.

Filosofia de Operação

Containers são Efêmeros

Containers devem ser tratados como **descartáveis**.

Nenhuma informação importante deve existir apenas dentro de um container.
Tudo que precisa persistir deve estar em:

- Bancos de dados
- Redis
- Volumes NFS
- Storage externo
- Serviços de logs
- Sistemas de métricas

Se um container puder ser destruído e recriado sem perda de dados, então o sistema está corretamente arquitetado.

Deploy é Versionado

Toda aplicação publicada no cluster deve possuir:

- Repositório Git
- Versionamento semântico
- Imagem Docker versionada
- Deploy via stack
- Histórico de versões
- Possibilidade de rollback

O cluster não deve rodar código sem versão identificável.

Isso permite:

- Rastreabilidade
- Auditoria
- Rollback rápido
- Reprodutibilidade de ambiente
- Comparação entre versões

Um Processo por Container

O cluster adota o princípio de:

> Um container deve executar apenas uma responsabilidade.

Isso melhora:

- Isolamento de falhas
- Escalabilidade independente
- Observabilidade
- Logging
- Atualizações
- Diagnóstico de problemas
- Consumo de recursos previsível

Exemplo de separação comum:

Responsabilidade	Serviço
Web	nginx
Aplicação	php-fpm
Scheduler	schedule
Worker	queue
Migration	migrate
Seeder	seeder

Essa separação é parte fundamental da arquitetura do cluster.

Filosofia de Operação

Logs e Métricas são Obrigatórios

Todo serviço deve gerar logs e métricas.

Se um serviço não gera logs, ele não pode ser operado corretamente.
Se não existem métricas, não existe monitoramento real.

A observabilidade não é opcional, ela faz parte da plataforma.

Automação antes de operação manual

Sempre que possível, tarefas devem ser automatizadas.

Evitar:

- Alterações manuais em containers
- Alterações manuais em servidores
- Configurações feitas apenas via interface gráfica
- Deploy manual sem versionamento
- Execução manual de procedimentos repetitivos

Preferir:

- Scripts
- CI/CD
- Stack versionada
- Dockerfile
- Infraestrutura como código
- Secrets do Swarm
- Configs do Swarm

O objetivo é que qualquer ambiente possa ser recriado do zero apenas com código e configurações versionadas.

Simplicidade Operacional

Uma das decisões arquiteturais do cluster foi priorizar **simplicidade operacional**.

Isso significa:

- Menos ferramentas quando possível
- Menos camadas de abstração
- Sistemas que a equipe entende completamente
- Troubleshooting simples
- Menor overhead operacional
- Menor curva de aprendizado
- Menor dependência de especialistas

A complexidade é considerada um custo operacional.

Padronização

Aplicações publicadas no cluster devem seguir padrões definidos para:

- Estrutura de containers
- Deploy
- Versionamento
- Uso de secrets
- Uso de Redis
- Uso de volumes
- Logs
- Redes
- Exposição via Traefik
- WAF
- Observabilidade

Padronização reduz erros, facilita troubleshooting e permite que qualquer pessoa da equipe opere qualquer serviço.

Responsabilidade do Cluster vs Responsabilidade da Aplicação

Divisão clara de responsabilidades:

Cluster	Aplicação
Rede	Regra de negócio
Proxy	Código
TLS	Queries
Logs	Validação
Métricas	Processamento
Storage	Modelos
Autenticação	Interface
Observabilidade	APIs
Deploy	Features

O cluster fornece a **infraestrutura e a plataforma**, a aplicação fornece a **lógica de negócio**.

Resumo da Filosofia

O cluster foi projetado para ser previsível, reproduzível, observável e operável.

Aplicações devem ser empacotadas como containers, implantadas de forma versionada, gerar logs, expor métricas e não depender de servidores específicos.

A plataforma deve abstrair a infraestrutura para que as equipes possam focar no desenvolvimento de software, enquanto o cluster fornece rede, segurança, storage, observabilidade e mecanismos de deploy.

A simplicidade operacional, a padronização e a automação são princípios fundamentais da operação do ambiente.