

DATA SHEET · IMPLANTAÇÃO

Implantação da plataforma GitLab™ em Kubernetes

GitLab self-managed pelo GitLab Helm chart ou pelo GitLab Operator, aplicados pelo GitLab Environment Toolkit em cluster Kubernetes existente do cliente ou, no Cloud Native Hybrid na AWS, em EKS criado pelo pipeline, nas arquiteturas Cloud Native e Cloud Native Hybrid

CÓDIGO**PS-IMP-02****TIPO****Data sheet****VERSÃO****1.2****DATA****1 de outubro de 2026**

O3S Consultoria e Tecnologia da Informação Ltda.
contato@pointertech.digital · +55 (61) 99874-4648 · pointertech.digital

GITLAB is a trademark of GitLab Inc. in the United States and other countries and regions.

**Select
Partner**

DATA SHEET • PS-IMP-02

Implantação da plataforma GitLab em Kubernetes

GitLab self-managed pelo GitLab Helm chart ou pelo GitLab Operator, aplicados pelo GitLab Environment Toolkit em cluster Kubernetes existente do cliente ou, no Cloud Native Hybrid na AWS, em EKS criado pelo pipeline, nas arquiteturas Cloud Native e Cloud Native Hybrid

Implantação de uma instância GitLab self-managed em **cluster Kubernetes** do cliente, em data center próprio ou em nuvem, pelo **GitLab Helm chart** oficial ou pelo **GitLab Operator**, aplicados pelo **GitLab Environment Toolkit (GET)** sem modificação. O cluster pode já existir ou, no Cloud Native Hybrid na AWS, ser criado pelo pipeline (EKS e VMs de backend) com os módulos Terraform oficiais do GET. Duas famílias de arquitetura de referência GitLab: **Cloud Native** (S, M, L e XL, até 1.000 RPS), com todos os componentes GitLab no cluster e PostgreSQL, Redis e object storage externos; e **Cloud Native Hybrid** (2.000 a 50.000 usuários), com Webservice, Sidekiq e serviços de suporte no cluster e Gitaly, PostgreSQL e Redis em VMs ou serviços externos. Antes de qualquer instalação, o pipeline verifica o cluster e testa os serviços externos de dentro do próprio cluster.

1. Para quem

- Organizações com cluster Kubernetes existente, em data center próprio ou gerenciado em nuvem, que vão operar a plataforma GitLab nesse cluster.
- Organizações com conta AWS que contratam a criação do Cloud Native Hybrid pelo pipeline: VMs de backend e cluster EKS.
- Clientes com PostgreSQL, Redis e object storage S3-compatível externos ao cluster (serviços gerenciados de nuvem ou servidores próprios), exigidos pelo GitLab Helm chart.
- Ambientes dimensionados pelas arquiteturas de referência Cloud Native (até 100, 200, 500 ou 1.000 RPS) ou Cloud Native Hybrid (2.000 a 50.000 usuários).
- Ambientes que precisam de atualização de versão sem parada: Cloud Native Hybrid com backends em alta disponibilidade (3k ou mais) ou com PostgreSQL externo.
- Clientes que precisam de Container Registry na instância, de runners próprios no Kubernetes ou de clusters conectados à instância pelo agente GitLab para Kubernetes.
- Clientes com assinatura GitLab Premium ou GitLab Ultimate (o GitLab Environment Toolkit exige Premium ou superior).

2. Onde executamos

A implantação é feita em **cluster Kubernetes já existente**, acessado pelo pipeline com um kubeconfig de conta de serviço dedicada, ou, no **Cloud Native Hybrid na AWS**, em **cluster EKS e VMs de backend criados pelo pipeline** na conta do cliente com os módulos Terraform oficiais do GET, sem modificação (estado do Terraform no projeto do engajamento; o acesso ao EKS usa a identidade AWS do ambiente, sem kubeconfig entregue pelo cliente).

- **Data center próprio (on-premises):** distribuições conformes, como RKE2, OpenShift ou K3s, com provedor de Services do tipo LoadBalancer (por exemplo MetalLB ou ServiceLB) e StorageClass para os volumes do Gitaly.
- **Nuvem pública, cluster existente:** clusters gerenciados na conta do cliente (AKS, EKS, GKE, OKE), com PostgreSQL, Redis e object storage gerenciados onde a arquitetura permite. A documentação GitLab lista Amazon RDS, Google Cloud SQL e Azure Database for PostgreSQL Flexible Server como serviços PostgreSQL que funcionam, e classifica Amazon Aurora e Google AlloyDB como incompatíveis. Para Redis na Azure, a documentação GitLab informa que Azure Cache for Redis e Azure Managed Redis não oferecem Redis 7.2 nem versão suportada de Valkey e orienta Redis ou Valkey autogerenciado em VM.
- **Nuvem pública, cluster criado pelo pipeline:** Cloud Native Hybrid na AWS, com EKS (pools Webservice, Sidekiq e suporte), VMs de backend, rede e buckets. Sob consulta: AKS e GKE criados pelo pipeline (o módulo do GET para Azure cria somente VMs; clusters AKS ou GKE existentes entram como cluster existente) e Cloud Native (S a XL) com cluster criado pelo pipeline.
- **Cloud Native Hybrid:** além do cluster, VMs Linux para os componentes com estado (mesmos requisitos de VM da implantação em servidores Linux, PS-IMP-01) ou serviços gerenciados de PostgreSQL e Redis.
- **Rede:** o runner do engajamento fica em uma VM com acesso à API do cluster e somente conexões de saída (no EKS criado pelo pipeline, o endpoint da API é privado e o runner precisa alcançar a rede criada); os nós do cluster baixam as imagens de registry.gitlab.com e docker.io ou de um registry espelho. A operação sem acesso à internet (*air-gapped*) está no roadmap.

Segundo a documentação GitLab, a arquitetura Cloud Native aceita qualquer distribuição Kubernetes que atenda aos pré-requisitos do Helm chart, e rede, storage classes e autenticação do Kubernetes ficam fora do escopo do suporte da GitLab. Para a variante Cloud Native Hybrid, a documentação informa testes regulares em GKE e EKS.

3. Escopo do serviço

- Discovery de implantação e escolha da arquitetura de referência: Cloud Native S, M, L ou XL, ou Cloud Native Hybrid 2k a 50k (com Gitaly Cluster ou Sharded a partir de 3k), e da forma

de instalação (GitLab Helm chart ou GitLab Operator).

- Projeto do engajamento com a configuração do ambiente versionada (configuração como código), revisada por merge request e validada automaticamente pelo pipeline, incluindo cluster, serviços externos e dimensionamento dos pods.
- **Cloud Native Hybrid criado pelo pipeline na AWS, quando contratado:** VMs de backend, cluster EKS com os pools Webservice, Sidekiq e suporte, rede e buckets, pelos módulos Terraform oficiais do GET; plano com o resumo das ações antes de cada aplicação, aplicação aprovada no ambiente protegido e estado do Terraform no projeto do engajamento; aumento de pools por merge request.
- Instalação e registro do runner do engajamento em VM com acesso à API do cluster, execução em ambiente protegido e aprovação por etapa.
- Verificação prévia do cluster (versão do Kubernetes contra a matriz do GitLab Helm chart, permissão, StorageClass, nós e capacidade por grupo de nós) e sonda de PostgreSQL (versão, extensões e parâmetros), Redis e buckets S3 executada de dentro do cluster; no Hybrid, verificação prévia das VMs.
- Instalação pelo GitLab Helm chart na versão correspondente à versão GitLab do engajamento, ou pelo GitLab Operator (GitLab 17.6.0 ou superior), com Envoy Gateway (padrão) ou NGINX Ingress, Secrets do chart e, no Hybrid, Linux package nas VMs de backend.
- Configuração dos recursos declarados: TLS, object storage, GitLab Pages, Container Registry, busca avançada externa e escopos da busca global, LDAP, SMTP, licença, backups agendados, servidor do agente GitLab para Kubernetes (KAS) em kas. e Prometheus (kube-prometheus-stack instalado pelo GET).
- **Frota de runners do cliente, quando contratada:** runners criados na instância com escopo, tags e proteção definidos na configuração e instalados no Kubernetes pelo chart oficial gitlab-runner (inclusive no EKS criado pelo pipeline) ou em VMs, conferidos online ao final.
- **Agentes GitLab para Kubernetes, quando contratados:** agente registrado e instalado em cada cluster do cliente (inclusive o cluster da própria instância), com o acesso de CI/CD dos grupos e projetos autorizados e relatório das conexões.
- Verificação pós-implantação automática e documento as-built gerado pelo pipeline.
- Rotinas de dia 2 disponíveis no pipeline: diagnóstico somente leitura, reexecução das migrations do banco, backup sob demanda, verificação avulsa e atualização de versão sem parada no Cloud Native Hybrid com backends em alta disponibilidade ou PostgreSQL externo.
- Transferência de conhecimento e entrega do repositório do engajamento.

4. Arquiteturas suportadas

- **Disponível** Ofertado nas condições deste documento.

- **Sob consulta** Escopo e condições definidos em proposta específica, com execução piloto.

Arquitetura	Carga de referência	Componentes	Estado
Cloud Native S	até 100 RPS; carga leve, não adequada a monorepos em uso ativo	No cluster: Webservice 6 a 9 pods, Sidekiq 8 a 12 pods, Gitaly 3 pods (um nó dedicado por pod), serviços de suporte 12 vCPU / 48 GB. Externos: PostgreSQL 8 vCPU / 32 GB, Redis cache e Redis persistente 2 vCPU / 8 GB cada, object storage	• Disponível
Cloud Native M	até 200 RPS; carga moderada, monorepos de uso leve	No cluster: Webservice 14 a 21 pods, Sidekiq 16 a 24 pods, Gitaly 3 pods (15 vCPU / 62 GB cada), suporte 12 vCPU / 48 GB. Externos: PostgreSQL 16 vCPU / 64 GB, Redis cache e persistente 2 vCPU / 8 GB cada, object storage	• Disponível
Cloud Native L	até 500 RPS; carga alta, monorepos de uso moderado	No cluster: Webservice 28 a 42 pods, Sidekiq 32 a 48 pods, Gitaly 3 pods (31 vCPU / 126 GB cada), suporte 12 vCPU / 48 GB. Externos: PostgreSQL 32 vCPU / 128 GB, Redis cache e persistente 2 vCPU / 16 GB cada, object storage	• Disponível
Cloud Native XL	até 1.000 RPS; carga intensa, monorepos de uso intenso	No cluster: Webservice 56 a 84 pods, Sidekiq 64 a 96 pods, Gitaly 3 pods (63 vCPU / 254 GB cada), suporte 24 vCPU / 96 GB. Externos: PostgreSQL 64 vCPU / 256 GB, Redis cache e persistente 2 vCPU / 16 GB cada, object storage	• Disponível
Cloud Native Hybrid 2k	até 40 RPS / 2.000 usuários	No cluster: Webservice, Sidekiq e suporte (12 + 3,6 + 4 vCPU solicitados). Em VMs: 1 Gitaly; PostgreSQL e Redis em 1 VM cada ou em serviços externos	• Disponível
Cloud Native Hybrid 3k	até 60 RPS / 3.000 usuários	No cluster: Webservice, Sidekiq e suporte (16 + 7,2 + 4 vCPU). Em VMs: 3 Consul, 3 PostgreSQL, 3 PgBouncer, balanceador interno, 3 Redis com Sentinel, 3 Gitaly, 3 Praefect e 1 PostgreSQL do Praefect (20 VMs com Gitaly Cluster)	• Disponível
Cloud Native Hybrid 5k	até 100 RPS / 5.000 usuários	No cluster: 36 + 7,2 + 4 vCPU. Em VMs: mesmos papéis do Hybrid 3k, com PostgreSQL e Gitaly maiores (20 VMs com Gitaly Cluster)	• Disponível
Cloud Native Hybrid 10k	até 200 RPS / 10.000 usuários	No cluster: 80 + 12,6 + 8 vCPU. Em VMs: papéis do Hybrid 3k com Redis separado em cache e persistente (23 VMs com Gitaly Cluster)	◦ Sob consulta
Cloud Native Hybrid 25k	até 500 RPS / 25.000 usuários	No cluster: 140 + 12,6 + 8 vCPU. Em VMs: papéis do Hybrid 10k com nós maiores (23 VMs com Gitaly Cluster)	◦ Sob consulta
Cloud Native Hybrid 50k	até 1.000 RPS / 50.000 usuários	No cluster: 308 + 12,6 + 8 vCPU. Em VMs: papéis do Hybrid 10k com nós maiores (23 VMs com Gitaly Cluster)	◦ Sob consulta

- Carga de referência conforme a documentação GitLab. Cloud Native é dimensionada só por RPS; a equivalência em usuários vale para Linux package e Cloud Native Hybrid (para cada 1.000 usuários, a GitLab testa 20 RPS de API, 2 RPS web, 2 RPS de Git pull e 0,4 RPS de Git push).

- Cloud Native: segundo a documentação GitLab, o Gitaly no Kubernetes funciona somente no modo sharded, e cada pod de Gitaly é ponto único de falha para os repositórios que atende; o Gitaly Cluster (Praefect) no Kubernetes está em beta e não faz parte da arquitetura de referência.
- PostgreSQL e Redis dentro do Kubernetes não são suportados pelas arquiteturas de referência GitLab; o pipeline exige PostgreSQL e Redis externos na Cloud Native e aceita VMs ou serviços externos no Hybrid.
- Os totais de vCPU e memória dos grupos de nós cobrem só os componentes GitLab; os processos de sistema do Kubernetes exigem recursos adicionais (documentação GitLab). Detalhamento por arquitetura no documento de pré-requisitos PR-IMP-02.
- A arquitetura de referência 1k não tem variante Cloud Native Hybrid.
- Cloud Native Hybrid 2k: PostgreSQL, Redis e Gitaly em nó único, sem alta disponibilidade. A atualização de versão sem parada exige backends em alta disponibilidade (Patroni e Consul, 3k ou mais) ou PostgreSQL externo; no Hybrid 2k com PostgreSQL em VM e na Cloud Native, a atualização usa janela de manutenção.
- As arquiteturas Cloud Native Hybrid valem para cluster e VMs existentes e para EKS e VMs criados pelo pipeline na AWS; no EKS criado pelo pipeline, a quantidade de nós de cada pool é calculada pelos totais da arquitetura ou definida na configuração.
- Cloud Native S, M, L e XL usam os mesmos componentes e papéis, com outra quantidade e tamanho de nós e de réplicas.
- Arquiteturas em estado *Sob consulta* (Cloud Native Hybrid 10k a 50k): escopo e condições definidos em proposta específica, com execução piloto.

5. Recursos configurados

- Nativo — recurso da plataforma GitLab
 - Requer configuração
- Integração externa — depende de serviço do cliente ou de terceiro

Recurso	Classificação	Detalhe
Cloud Native Hybrid criado pelo pipeline (AWS)	◦ Integração externa	Depende da conta AWS do cliente e de uma identidade com permissão para criar os recursos, inclusive o provedor OpenID Connect do IAM que o módulo EKS do GET sempre cria. Módulo Terraform oficial do GET 3.11.0, sem modificação: VMs de backend, cluster EKS em versão da matriz do chart com endpoint da API privado, pools Webservice, Sidekiq e suporte, rede e buckets. Estado no GitLab-managed Terraform state do projeto do engajamento; kubeconfig gerado a partir do estado. Registros DNS e certificados não são criados pelo pipeline. AKS e GKE criados pelo pipeline e Cloud Native com cluster criado pelo pipeline: sob consulta.

Recurso	Classificação	Detalhe
GitLab Operator	• Requer configuração	Alternativa ao Helm chart, com os mesmos values (GitLab 17.6.0 ou superior): o GET instala cert-manager, Envoy Gateway, o GitLab Operator e o recurso GitLab, e a verificação pós-implantação confere a fase do recurso GitLab. O GET classifica o suporte ao Operator como Beta, e a documentação GitLab registra limitações conhecidas do Operator; a atualização segue o método sem parada, uma versão menor por vez. A troca do Helm chart pelo Operator recria o balanceador do Envoy Gateway com endereço novo: os registros DNS são atualizados em janela de manutenção. GitLab Operator em OpenShift: sob consulta.
Envoy Gateway	• Nativo	Gateway API com Envoy Gateway embutido no GitLab Helm chart, padrão desde a versão 19.0, instalado pelo GET. Expõe HTTPS e o SSH do Git (porta padrão 22) por um Service do tipo LoadBalancer; anotações do LoadBalancer (por exemplo IP fixo) configuráveis. Listeners da instância GitLab, do Container Registry, do GitLab Pages e do KAS com o certificado do cliente.
NGINX Ingress	• Requer configuração	Opção alternativa, com IP externo obrigatório. Segundo a documentação do chart, está deprecated e disponível até ser removido na versão 20.0; o pipeline emite alerta ao usá-lo.
Provedor de LoadBalancer do cluster	○ Integração externa	Services do tipo LoadBalancer providos pela nuvem ou, em data center, por soluções como MetalLB ou ServiceLB. Em RKE2, K3s, OpenShift ou outra distribuição, o preflight emite alerta para confirmar esse provedor. O balanceador externo do cliente no lugar do LoadBalancer do Gateway não é suportado no Kubernetes.
TLS com certificado do cliente	• Requer configuração	Certificado e chave do host externo publicados como Secret usado pelo Gateway. Opção padrão. O certificado cobre também os nomes do Container Registry e do KAS (kas.); com autoridade certificadora não pública, a cadeia da AC emissora é instalada como confiável nos nós com Linux package do Hybrid.
TLS com Let's Encrypt	○ Integração externa	Emissão pela autoridade Let's Encrypt por meio do cert-manager, com e-mail de registro obrigatório.

Recurso	Classificação	Detalhe
PostgreSQL externo	○ Integração externa	Serviço gerenciado ou servidor do cliente, PostgreSQL 17 para GitLab 19.x. Com usuário administrador, o GET cria usuário, banco e extensões; sem ele, esses itens precisam existir antes. A sonda confere versão, extensões e <code>max_locks_per_transaction</code> (erro) e os parâmetros exigidos pela documentação GitLab para banco externo (alerta). Obrigatório na Cloud Native; no Hybrid, alternativa às VMs de PostgreSQL.
Redis externo	○ Integração externa	Redis 7.0 ou superior (a documentação GitLab cita 7.0 como mínima e 7.2 como versão de referência; Valkey a partir de 7.2), instância standalone com ou sem alta disponibilidade (Redis Cluster e variantes serverless não são suportados), com autenticação e TLS opcional. Obrigatório na Cloud Native; no Hybrid, alternativa às VMs de Redis.
Object storage S3-compatível	○ Integração externa	Obrigatório. Um bucket por tipo de objeto (13 buckets, incluindo backups e o bucket temporário de restauração, e o bucket do Container Registry quando habilitado), conferidos pela sonda antes da implantação. Segundo a documentação do chart, instalações Helm exigem buckets separados por tipo, senão a restauração de backup falha.
Gitaly no cluster (Cloud Native)	• Requer configuração	StatefulSet com um volume persistente por pod (padrão 50 GiB), na StorageClass informada pelo cliente, e um nó dedicado por pod.
Componentes com estado em VMs (Cloud Native Hybrid)	• Nativo	Gitaly Cluster (Praefect) ou Gitaly Sharded, PostgreSQL com Patroni, PgBouncer, Consul e Redis com Sentinel instalados pelo GET com o Linux package nas VMs, conforme a arquitetura.
Autoescalamento de Webservice e Sidekiq	• Nativo	Réplicas mínimas e máximas definidas pela arquitetura de referência (na Cloud Native, mínimo em torno de 2/3 do máximo, segundo a documentação GitLab); ajuste abaixo da arquitetura gera alerta.
GitLab Pages	• Requer configuração	URL raiz com DNS wildcard, certificado wildcard com HTTPS e bucket próprio no object storage.

Recurso	Classificação	Detalhe
Container Registry	• Requer configuração	Container Registry da instância em subdomínio (por exemplo registry.), exposto pelo Envoy Gateway com o certificado do cliente, que precisa cobrir esse nome; storage no object storage S3 (bucket próprio). Com o registry desabilitado, os backups do toolbox excluem o componente registry.
Busca avançada (Advanced Search)	◦ Integração externa	Cluster Elasticsearch ou OpenSearch externo do cliente. Na Cloud Native, somente modo externo; no Hybrid, também VMs OpenSearch instaladas pelo GET.
Escopos da busca global	• Requer configuração	Liga ou desliga os escopos da busca global da instância (código, commits, wiki, itens de trabalho, merge requests, usuários, grupos e títulos de snippets), aplicados pelo pod toolbox e lidos de volta na implantação. Código, commits e wiki em toda a instância exigem a busca avançada (GitLab Premium ou GitLab Ultimate). Busca exata de código (Zoekt): sob consulta.
LDAP / Active Directory e SMTP	◦ Integração externa	Servidores do cliente; as senhas são publicadas como Secrets do chart pelo pipeline.
Backups agendados	• Requer configuração	CronJob do toolbox executando o backup-utility para o bucket de backups, padrão diário às 02:00; a retenção em dias vira quantidade máxima de backups (agenda não diária gera alerta).
Licença	• Requer configuração	Activation code (ativação online) ou arquivo de licença (offline), aplicados pelo GET.
Agente GitLab para Kubernetes	• Requer configuração	Servidor do agente (KAS) habilitado por padrão em kas., pelo mesmo balanceador da instância GitLab, com o certificado do cliente; a verificação pós-implantação confere a resposta do KAS pela URL externa. Agentes conectados na entrega (opcional): config.yaml com o acesso de CI/CD dos grupos e projetos autorizados, registro e token pela API e agentk instalado pelo chart oficial em cada cluster do cliente; relatório das conexões.

Recurso	Classificação	Detalhe
Frota de runners do cliente	• Requer configuração	Runners criados na instância pela API, com escopo de instância, grupo ou projeto, tags, proteção, execução sem tag e timeout máximo definidos na configuração, instalados no Kubernetes pelo chart oficial gitlab-runner (no EKS criado pelo pipeline, com o acesso lido do estado do Terraform) ou em VMs (executor Docker, docker-autoscaler na AWS). Executor shell e docker-autoscaler no Azure: sob consulta.
Monitoramento (Prometheus)	• Requer configuração	kube-prometheus-stack (chart da comunidade Prometheus) instalado pelo GET no namespace monitoring, com coleta do chart e, no Hybrid, das VMs de backend; volume persistente com tamanho e StorageClass definidos na configuração (no EKS criado pelo pipeline, StorageClass gp3 criada pela implantação). Padrão a partir desta versão; pode ser desligado quando o cliente tem monitoramento próprio. A verificação pós-implantação confere os alvos ativos. Grafana gerenciado pela Pointer, com o Prometheus da instância como fonte de dados e login pela instância GitLab: sob consulta.
Atualização de versão	• Requer configuração	Nova versão definida por merge request, respeitando o caminho oficial de upgrade e a matriz de versões do Kubernetes, com backup sob demanda antes. Cloud Native Hybrid com backends em alta disponibilidade (Patroni e Consul, 3k ou mais) ou com PostgreSQL externo: atualização sem parada pelo playbook de zero downtime do GET, com 2 ou mais réplicas de Webservice e Sidekiq (a documentação GitLab prevê atualização sem parada em Cloud Native Hybrid desde a 18.9). Hybrid 2k com PostgreSQL em VM e Cloud Native: nova execução da implantação em janela de manutenção.

6. Ferramentas

FERRAMENTA OFICIAL DA GITLAB

GitLab Environment Toolkit (GET)

Imagem oficial do GET 3.11.0, sem modificação. Instala o Envoy Gateway e aplica o GitLab Helm chart (ou o GitLab Operator) com os values gerados pelo pipeline, mesclados por Custom Config; os Secrets de object storage, backup, LDAP, SMTP e Container Registry são criados por Custom Tasks. No Hybrid, instala o Linux package nas VMs de backend a partir de inventário estático; instala o kube-prometheus-stack e executa o playbook de zero downtime. No Hybrid criado pelo pipeline na AWS, o módulo Terraform oficial do GET cria as VMs de backend e o EKS.

CHART OFICIAL DA GITLAB

GitLab Helm chart

Versão escolhida pelo GET a partir da versão GitLab do engajamento (por exemplo, GitLab 19.4.1 corresponde ao chart 10.4.1). Release sempre com o nome gitlab, no namespace definido no engajamento.

OPERADOR OFICIAL DA GITLAB

GitLab Operator

Opção à instalação pelo Helm chart: o recurso GitLab recebe os mesmos values gerados pelo pipeline; sem release Helm, a verificação usa a fase do recurso GitLab.

FERRAMENTA HASHICORP, NA VERSÃO FIXADA PELO GET

Terraform

Somente no Hybrid criado pelo pipeline: executa o módulo AWS do GET com o estado no GitLab-managed Terraform state do projeto do engajamento; antes de aumentar um pool do EKS, o pipeline ajusta a capacidade desejada dos grupos de nós existentes.

PACOTE OFICIAL DA GITLAB

Linux package

Somente no Cloud Native Hybrid: instalado pelo GET nas VMs de Gitaly, Praefect, PostgreSQL, PgBouncer, Consul e Redis que não forem substituídas por serviços externos.

FERRAMENTAS INCLUÍDAS NA IMAGEM OFICIAL DO GET

Ansible, kubectl e Helm

Executam os playbooks do GET, a coleta do estado do cluster, a sonda, a verificação pós-implantação, o diagnóstico e a instalação dos charts da frota de runners e dos agentes.

FERRAMENTA POINTER**psctl**

Valida a configuração do engajamento (cluster, serviços externos e dimensionamento dos pods), gera o ambiente do GET, os values do chart e o módulo Terraform, lê os endereços e o kubeconfig do estado do Terraform, lista as credenciais exigidas, avalia a capacidade do cluster, executa a sonda dos serviços externos, cria e verifica os runners e os agentes pela API e gera o as-built.

FERRAMENTA OFICIAL DA GITLAB**GitLab Runner**

Runner do engajamento: instalado em VM com executor Docker, registrado somente no projeto do engajamento, protegido e sem jobs sem tag; acessa a API do cluster com o kubeconfig do engajamento (no EKS criado pelo pipeline, gerado a partir do estado do Terraform) e, no Hybrid, as VMs por SSH; faz somente conexões de saída e é removido no encerramento (ou transferido ao cliente). Frota do cliente: chart oficial gitlab-runner no Kubernetes ou pacote oficial nas VMs.

CHART OFICIAL DA GITLAB (GITLAB-AGENT)**Agente GitLab para Kubernetes (agentk)**

Instalado em cada cluster do cliente com o endereço e a versão do KAS publicados pela instância; o acesso dos pipelines ao cluster segue o config.yaml versionado no projeto de configuração do agente.

7. Como o pipeline Pointer executa



Etapas sob demanda: Diagnóstico do cluster (somente leitura) (**diagnostico**); Reexecução das migrations do banco (**migrations-reexecutar**); Backup sob demanda (**backup-agora**); Verificação sob demanda (**verificar-agora**); Atualização de versão sem parada (Cloud Native Hybrid) (**upgrade-zdu**).

- 1. Criação da infraestrutura (opcional, Cloud Native Hybrid na AWS)** — a configuração gera o módulo Terraform em runner hospedado, sem credenciais; o plano mostra o resumo das ações e a aplicação, aprovada no ambiente protegido, cria VMs de backend, EKS, rede e buckets. A destruição só é executada com decisão registrada por merge request.
- 2. Validação da configuração** — em runner hospedado, sem credenciais e sem acesso à rede do cliente: confere versão, URL externa, arquitetura, seção do cluster, serviços externos e

dimensionamento dos pods; gera os values do chart, as tarefas que criam os Secrets e a lista de credenciais exigidas.

3. **Verificação prévia do cluster e dos serviços externos** — no runner do engajamento: versão do Kubernetes contra a matriz do chart, permissão cluster-admin, StorageClass, nós prontos e capacidade por grupo de nós; um Job temporário no namespace testa PostgreSQL, Redis e os buckets S3 de dentro do cluster e é removido ao final. No Hybrid, verifica também as VMs por SSH.
4. **Implantação no cluster** — com aprovação de pessoa diferente de quem executa: repete a verificação prévia (qualquer erro impede o início) e executa o GET com o Helm chart ou o GitLab Operator. Em falha, grava um diagnóstico somente leitura do cluster. Uma execução por ambiente de cada vez; tempo máximo de 4 horas.
5. **Verificação pós-implantação** — automática: release Helm implantado (com o GitLab Operator, fase do recurso GitLab), deployments disponíveis com rollout concluído, pods em execução, versão, checagem do Gitaly, página de login pela URL externa, KAS em kas. e alvos ativos do Prometheus; no Hybrid, também Patroni e Praefect nas VMs.
6. **Documento as-built** — com distribuição, namespace, ingress, StorageClass, réplicas e recursos por componente e serviços externos.

Componentes opcionais no mesmo projeto: frota de runners do cliente e agentes GitLab para Kubernetes (validação, aplicação e verificação). Rotinas de dia 2 disponíveis no mesmo pipeline: diagnóstico do cluster (somente leitura), reexecução das migrations do banco (instalação pelo Helm chart), backup sob demanda, verificação avulsa e atualização de versão sem parada no Cloud Native Hybrid com backends em alta disponibilidade ou PostgreSQL externo.

8. Metodologia

Assessment → Implantação → Otimização → Transferência de conhecimento.

Fase 1**Assessment**

- Kick-off: escopo, papéis, janelas de manutenção, acessos ao cluster, aprovadores das execuções em produção e critérios de aceite.
- Discovery Kubernetes: distribuição, versão e administração do cluster; cluster dedicado ou compartilhado, namespace, cotas e políticas (ResourceQuota, LimitRange, Pod Security, OPA ou Kyverno), grupos de nós, StorageClass, LoadBalancer e DNS (gitlab, registry e kas), saída dos nós para os registries, PostgreSQL, Redis e object storage, TLS, LDAP, SMTP, backup e licenciamento.
- Definição da forma da infraestrutura (cluster existente ou Cloud Native Hybrid criado pelo pipeline na AWS), da instalação (Helm chart ou GitLab Operator) e dos recursos opcionais: Container Registry, escopos da busca global, monitoramento, frota de runners e clusters a conectar pelo agente GitLab para Kubernetes.
- Escolha da arquitetura de referência (Cloud

Fase 2**Implantação**

- Cloud Native Hybrid criado pelo pipeline: plano revisado com o cliente e aplicação aprovada no ambiente protegido.
- Instalação do runner do engajamento com acesso à API do cluster, ambiente protegido com aprovação e cadastro das credenciais, incluindo o kubeconfig da conta de serviço dedicada (dispensado no EKS criado pelo pipeline).
- Verificação prévia do cluster e sonda dos serviços externos; correção dos erros e registro dos alertas aceitos no registro de decisões.
- Implantação aprovada e executada pelo pipeline, com a verificação prévia repetida de forma bloqueante.
- Verificação pós-implantação automática e geração do as-built.
- Frota de runners e agentes GitLab para Kubernetes: criação na instância, instalação e verificação pelos componentes do pipeline.

Fase 3**Otimização**

- Execução do CronJob de backup ao menos uma vez (ou backup sob demanda), com o backup presente no bucket.
- Validação com o cliente do clone por HTTPS e SSH, do login LDAP e do e-mail de teste, quando habilitados.
- Tratamento de falhas pelas rotinas do pipeline: diagnóstico somente leitura, verificação avulsa e reexecução das migrations do banco.
- Ajustes de configuração por merge request e nova execução do pipeline; operação assistida no período acordado na proposta (oferta PS-OPS-01).

Fase 4**Transferência de conhecimento**

- Revisão do as-built com a equipe do cliente e tratamento das pendências registradas nele: exportação do Secret gitlab-rails-secrets para fora do cluster e troca da senha inicial do root.
- Sessões sobre a arquitetura implantada, o pipeline e as rotinas de dia 2 (backup, diagnóstico, verificação avulsa, migrations e atualização de versão, inclusive sem parada quando aplicável).
- Entrega do repositório do engajamento, com opção de espelhamento na instância GitLab do cliente.
- Cloud Native Hybrid criado pelo pipeline: a transferência do estado do Terraform é definida com o cliente e registrada no engajamento; infraestrutura temporária é destruída somente por decisão registrada em merge request.
- Encerramento: runner do engajamento removido ou transferido ao cliente, kubeconfig e demais credenciais revogados, projeto arquivado, termo de aceite e

Native S a XL ou Cloud Native Hybrid 2k a 50k). - Envio do documento de pré-requisitos PR-IMP-02 e acompanhamento do checklist de prontidão. - Criação do projeto do engajamento, configuração por merge request revisado por outro consultor e validação automática, com a lista das credenciais exigidas.			relatório de encerramento.
---	--	--	----------------------------

9. Entregáveis

Entregável	Descrição	Formato
Configuração do engajamento	Configuração do ambiente (arquitetura, cluster ou infraestrutura a criar, serviços externos, VMs no Hybrid, recursos, frota de runners e agentes), registro de decisões e runbook, versionados e revisados por merge request. Não contém credenciais.	Repositório Git
Plano e registro da infraestrutura criada	Quando o pipeline cria o Cloud Native Hybrid na AWS: plano do Terraform com o resumo das ações, registro da aplicação e endereços internos por papel lidos do estado do Terraform, que fica no projeto do engajamento.	Texto e JSON (artefatos do pipeline); estado do Terraform no projeto
Relatório de verificação prévia do cluster	Nós, capacidade exigida por grupo de nós, resultado da sonda de PostgreSQL, Redis e object storage, erros e alertas; no Hybrid, também o relatório por VM.	Markdown e JSON (artefatos do pipeline)
Relatório de verificação pós-implantação	Resultado de cada checagem (OK ou FALHOU): release Helm ou recurso do GitLab Operator, deployments e rollout, pods, versão, Gitaly, URL externa, KAS e Prometheus; no Hybrid, também as checagens das VMs.	JSON (artefato do pipeline)
Diagnóstico do cluster	Gerado automaticamente quando a implantação ou a atualização sem parada falha, ou sob demanda: nós, objetos do namespace, eventos, logs dos pods com problema e das migrations. Somente leitura.	Texto (artefato do pipeline)

Entregável	Descrição	Formato
As-built	Cliente, contrato, versões GitLab, do chart e do GET, distribuição, namespace, ingress, StorageClass, réplicas e recursos por componente, serviços externos, buckets, nomes (sem valores) das credenciais, resultado das checagens e pendências de entrega.	Markdown (artefato do pipeline, retido por 1 ano)
Relatório da frota de runners	Quando contratada: estado de cada runner na instância (identificador, status, tags, proteção, timeout e versão dos managers).	Markdown e JSON (artefatos do pipeline, retidos por 1 ano)
Relatório dos agentes GitLab para Kubernetes	Quando contratados: conexões de cada agente registradas na instância.	Markdown e JSON (artefatos do pipeline, retidos por 1 ano)
Evidência de backup	Registro da execução do CronJob de backup ou do backup sob demanda, com o backup presente no bucket de backups.	Registro no projeto do engajamento
Termo de aceite e relatório de encerramento	Aceite pelos critérios do engajamento (checagens obrigatórias da verificação OK, URL externa com certificado válido, clone por HTTPS e SSH, login LDAP e e-mail de teste validados quando habilitados, backup no bucket, runners da frota online e agentes conectados quando contratados) e registro do encerramento.	Documento

10. Premissas

- Cluster Kubernetes existente, administrado pelo cliente, em versão suportada pelo GitLab Helm chart e com capacidade conforme a arquitetura escolhida e o documento de pré-requisitos, ou, no Cloud Native Hybrid na AWS, EKS e VMs de backend criados pelo pipeline.
- Kubeconfig de conta de serviço dedicada ao engajamento com permissão cluster-admin, gerado com o administrador do cluster e revogado no encerramento (o GET cria namespace, CRDs do Gateway API e recursos de escopo de cluster). No EKS criado pelo pipeline, o acesso usa a identidade AWS do ambiente, com acesso ao cluster.
- Cloud Native Hybrid criado pelo pipeline: identidade AWS do cliente com permissão para criar os recursos do GET, inclusive o provedor OpenID Connect do IAM, sem políticas da organização que neguem essas ações, cotas da região suficientes e runner com rota para a rede criada (endpoint da API do EKS privado).

- PostgreSQL, Redis e object storage S3-compatível externos ao cluster, providos pelo cliente (na Cloud Native, obrigatórios).
- Registros DNS de gitlab, registry e kas do domínio apontando para o LoadBalancer do Gateway, criados pelo cliente; na troca do Helm chart pelo GitLab Operator, o endereço do LoadBalancer muda e os registros são atualizados.
- Assinatura GitLab Premium ou GitLab Ultimate para o go-live.
- Instalação online: os nós do cluster baixam as imagens de registry.gitlab.com e docker.io (ou de registry espelho configurado no cluster) e o runner acessa gitlab.com, registry.gitlab.com, charts.gitlab.io e docker.io por HTTPS, diretamente ou por proxy. Com o Prometheus do pipeline, também prometheus-community.github.io (chart) e quay.io, registry.k8s.io e docker.io (imagens).
- Aprovação de cada implantação e de cada aplicação de infraestrutura por pessoa diferente de quem executa, em ambiente protegido; mudanças de configuração, de versão e de infraestrutura somente por merge request.
- Tempos e cronograma do engajamento definidos na proposta.

Fora do escopo

- Administração do cluster Kubernetes existente do cliente.
- Criação de AKS ou GKE pelo pipeline e Cloud Native (S a XL) com cluster criado pelo pipeline: sob consulta (clusters existentes nesses provedores entram como cluster existente).
- GitLab Operator em OpenShift: sob consulta.
- GitLab Geo no Kubernetes (Cloud Native Hybrid e Cloud Native): sob consulta.
- Atualização de versão sem parada na Cloud Native e no Cloud Native Hybrid sem backends em alta disponibilidade nem PostgreSQL externo: não suportada pelo pipeline (atualização com janela de manutenção).
- Busca exata de código (Zoekt): sob consulta.
- Frota de runners com executor shell ou com docker-autoscaler no Azure: sob consulta.
- Grafana gerenciado pela Pointer: sob consulta.
- Registros DNS e emissão de certificados: responsabilidade do cliente, inclusive no EKS criado pelo pipeline.
- Instalação sem acesso à internet (*air-gapped*): roadmap.
- PostgreSQL ou Redis dentro do cluster (não suportados pelas arquiteturas de referência GitLab).
- Autenticação SAML ou OIDC: não é configurada pelo pipeline nesta versão.
- Migração de dados de outra plataforma para a instância implantada (ofertas de migração PS-MIG-01 a PS-MIG-04).

Responsabilidades do cliente

- Disponibilizar o cluster Kubernetes em versão suportada, com nós rotulados por carga (ou pool único dimensionado), StorageClass e provedor de LoadBalancer, ou, no Hybrid criado pelo pipeline, fornecer a conta AWS, a identidade com as permissões e as cotas da região.
- Gerar o kubeconfig da conta de serviço dedicada com permissão cluster-admin (cluster existente).
- Prover PostgreSQL, Redis e object storage com os buckets criados (salvo buckets criados pelo pipeline na AWS), e as VMs de backend no Hybrid com cluster existente.
- Fornecer certificado TLS (ou optar por Let's Encrypt), registros DNS e licença GitLab Premium ou Ultimate.
- Disponibilizar a VM do runner e liberar as regras de rede (saída HTTPS, API do cluster, acesso dos pods aos serviços externos).
- Emitir os tokens de API da frota de runners e dos agentes e o acesso aos clusters, quando contratados.
- Designar patrocinador, ponto focal técnico, administrador do cluster, aprovador das execuções e janela de manutenção.
- Detalhamento no documento de pré-requisitos PR-IMP-02.

Detalhamento no documento de pré-requisitos PR-IMP-02, enviado após a escolha do serviço.

11. Duração

A duração depende da arquitetura e do dimensionamento do cluster, da forma da infraestrutura (existente ou criada pelo pipeline), dos recursos opcionais contratados, da prontidão dos pré-requisitos (cluster, serviços externos, DNS e certificados) e das janelas de mudança. Cada execução da implantação no cluster tem tempo máximo de 4 horas; cada aplicação da infraestrutura, de 2 horas; e a atualização sem parada, de 6 horas. O prazo do engajamento é definido na proposta.

12. Documentos relacionados

- PS-00 · Catálogo de Serviços
- PR-IMP-02 · Pré-requisitos: Implantação da plataforma GitLab em Kubernetes
- PS-IMP-01 · Implantação da plataforma GitLab em servidores Linux
- PS-OPS-01 · Operação assistida
- PS-ONB-01 · Onboarding do cliente na plataforma GitLab
- PS-MIG-01 · Migração GitLab para GitLab self-managed
- PS-CAP-01 · Capacitação na plataforma GitLab

O3S Consultoria e Tecnologia da Informação Ltda. · contato@pointertech.digital · +55 (61) 99874-4648 ·
pointertech.digital

PS-IMP-02 · versão 1.2 · 1 de outubro de 2026. GITLAB is a trademark of GitLab Inc. in the United States and other countries and regions. A Pointer é parceira GitLab Select.