

DATA SHEET · IMPLANTAÇÃO

Implantação da plataforma GitLab™ em servidores Linux

GitLab self-managed com Linux package, instalado pelo GitLab Environment Toolkit em máquinas virtuais existentes do cliente ou criadas pelo pipeline na conta AWS ou Azure do cliente, nas arquiteturas de referência de 1.000 a 50.000 usuários

CÓDIGO**PS-IMP-01****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-01

Implantação da plataforma GitLab em servidores Linux

GitLab self-managed com Linux package, instalado pelo GitLab Environment Toolkit em máquinas virtuais existentes do cliente ou criadas pelo pipeline na conta AWS ou Azure do cliente, nas arquiteturas de referência de 1.000 a 50.000 usuários

Implantação de uma instância GitLab self-managed em máquinas virtuais Linux do cliente, seguindo as **arquiteturas de referência GitLab** de 1.000 a 50.000 usuários (nó único, multi-nó e alta disponibilidade). As VMs podem já existir, em data center próprio ou em nuvem, ou ser criadas pelo pipeline na conta AWS ou Azure do cliente, com os módulos Terraform oficiais do **GitLab Environment Toolkit (GET)**. A instalação usa o **Linux package** e o GET oficiais, sem modificação, executados pelo pipeline Pointer a partir de um runner com acesso à rede das VMs. Cada execução é precedida de verificação automática dos servidores e seguida de verificação pós-implantação e documento as-built gerados pelo próprio pipeline. No mesmo engajamento, o pipeline também instala a frota de runners do cliente, conecta agentes GitLab para Kubernetes e configura um segundo site com GitLab Geo, quando contratados.

1. Para quem

- Organizações que vão operar uma instância GitLab self-managed em máquinas virtuais Linux próprias (qualquer hipervisor ou servidores físicos) ou em VMs já criadas na conta de nuvem do cliente.
- Organizações com conta AWS ou assinatura Azure que contratam a criação da infraestrutura da arquitetura de referência pelo pipeline: VMs por papel, rede e, na AWS, buckets do object storage.
- Ambientes dimensionados pelas arquiteturas de referência GitLab de 1.000 a 50.000 usuários (20 a 1.000 RPS de API).
- Ambientes que exigem alta disponibilidade: a documentação GitLab associa as arquiteturas em alta disponibilidade a 3.000 usuários ou mais.
- Ambientes com segundo site para GitLab Geo, balanceador de carga do próprio cliente ou Container Registry na instância.
- Clientes que precisam de runners próprios para os projetos de CI/CD ou de clusters Kubernetes conectados à instância pelo agente GitLab para Kubernetes.
- Clientes com assinatura GitLab Premium ou GitLab Ultimate (o GitLab Environment Toolkit exige Premium ou superior).
- Clientes que precisam de uma instância GitLab de destino para uma migração posterior executada pela Pointer.

2. Onde executamos

A implantação é feita sobre **infraestrutura já existente do cliente** ou sobre **infraestrutura criada pelo pipeline na conta AWS ou Azure do cliente**. Nos dois casos, a instalação recebe as máquinas virtuais por papel da arquitetura (inventário estático, acesso por SSH) e é executada a partir de um runner com acesso à rede das VMs.

- **Data center próprio (on-premises):** VMs em qualquer hipervisor (por exemplo VMware, Hyper-V, KVM) ou servidores físicos, com sistema operacional da matriz suportada, SSH e sudo.
- **Nuvem pública, VMs existentes:** VMs já criadas na conta do cliente (por exemplo AWS, Azure ou Google Cloud), tratadas como infraestrutura existente.
- **Nuvem pública, infraestrutura criada pelo pipeline (AWS e Azure):** o pipeline cria as VMs da arquitetura de referência com os módulos Terraform oficiais do GET, sem modificação. Na AWS, também a rede (VPC e subnets novas, ou uso de VPC e subnets existentes) e os buckets do object storage; no Azure, as VMs e a rede virtual, em resource group existente do cliente, com object storage S3-compatível existente. O estado do Terraform fica no projeto do engajamento, cada aplicação passa por plano e aprovação, e a destruição só ocorre por decisão registrada em merge request. A criação de infraestrutura no Google Cloud é atendida sob consulta; VMs existentes no Google Cloud entram como infraestrutura existente.
- **Rede:** o runner do engajamento fica em uma VM com rota para as VMs e faz somente conexões de saída (HTTPS para gitlab.com e registry.gitlab.com, SSH até as VMs). Na infraestrutura criada pelo pipeline, o runner fica na rede criada ou com rota até ela. As VMs instalam o Linux package a partir de packages.gitlab.com. A operação sem acesso à internet (*air-gapped*) está no roadmap.

Segundo a documentação GitLab, a plataforma roda em nuvem ou em infraestrutura própria desde que os requisitos sejam atendidos, e o suporte da GitLab cobre a plataforma GitLab, não a infraestrutura subjacente. Uma instância não pode abranger várias regiões geográficas; o GitLab Geo replica a instância para um segundo site.

3. Escopo do serviço

- Discovery de implantação e escolha da arquitetura de referência (1k a 50k), da forma da infraestrutura (existente ou criada pelo pipeline na AWS ou no Azure) e, em alta disponibilidade, do modo do Gitaly (Gitaly Cluster com Praefect ou Gitaly Sharded).
- Projeto do engajamento com a configuração do ambiente versionada (configuração como código), revisada por merge request e validada automaticamente pelo pipeline.
- **Infraestrutura criada pelo pipeline (AWS ou Azure), quando contratada:** VMs por papel da arquitetura de referência, rede e, na AWS, buckets do object storage, criados pelos módulos Terraform oficiais do GET; plano com o resumo das ações antes de cada aplicação.

aplicação aprovada no ambiente protegido, tags de custo e governança do cliente em todos os recursos e estado do Terraform no projeto do engajamento.

- Instalação e registro do runner do engajamento em VM com rota para as VMs, exclusivo do projeto, com execução em ambiente protegido e aprovação por etapa.
- Verificação prévia (preflight) dos servidores reais: sistema operacional e última versão GitLab prevista para ele, arquitetura de CPU, vCPU e memória contra a arquitetura de referência, espaço em disco, sincronismo de horário, DNS e instalação prévia.
- Instalação do Linux package na versão definida no engajamento pelo GitLab Environment Toolkit, em todas as VMs do inventário, incluindo balanceadores HAProxy do GET (2k ou mais) ou o balanceador do cliente no lugar deles, PostgreSQL com Patroni e PgBouncer, Redis com Sentinel, Consul e Gitaly Cluster ou Sharded (3k ou mais).
- Configuração dos recursos declarados: TLS, object storage S3-compatível, GitLab Pages, Container Registry, busca avançada e escopos da busca global, LDAP, SMTP, licença, backup agendado, servidor do agente GitLab para Kubernetes (KAS) e Prometheus no nó de monitoramento (2k ou mais).
- **GitLab Geo, quando contratado:** dois sites Linux package (primário e secundário), cada um implantado pelo pipeline, e configuração da replicação pelo playbook oficial do GET, executada a partir do site primário.
- **Frota de runners do cliente, quando contratada:** runners criados na instância com escopo (instância, grupo ou projeto), tags, proteção e timeout definidos na configuração, instalados em VMs (executor Docker), em autoscaling na AWS (docker-autoscaler) ou em Kubernetes (Helm), e conferidos online ao final.
- **Agentes GitLab para Kubernetes, quando contratados:** projeto de configuração com o acesso de CI/CD dos grupos e projetos autorizados, registro e token pela API e agente instalado em cada cluster do cliente, com relatório das conexões.
- Verificação pós-implantação automática e documento as-built gerado pelo pipeline.
- Rotinas de dia 2 disponíveis no pipeline: backup sob demanda, verificação avulsa e atualização de versão sem parada na alta disponibilidade (nós GitLab Rails dimensionados conforme a arquitetura de referência).
- 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
1k · nó único	até 20 RPS / 1.000 usuários	1 VM com todos os componentes (8 vCPU, 16 GB); object storage opcional	• Disponível
2k · multi-nó	até 40 RPS / 2.000 usuários	8 VMs: balanceador externo (HAProxy do GET ou do cliente), PostgreSQL, Redis, Gitaly, Sidekiq, 2 GitLab Rails e monitoramento; object storage obrigatório; sem alta disponibilidade	• Disponível
3k · alta disponibilidade	até 60 RPS / 3.000 usuários	27 VMs com Gitaly Cluster (23 com Gitaly Sharded): balanceadores externo e interno (HAProxy), 3 Consul, 3 PostgreSQL com Patroni, 3 PgBouncer, 3 Redis com Sentinel, 3 Gitaly, 3 Praefect e 1 PostgreSQL do Praefect, 2 Sidekiq, 3 GitLab Rails e monitoramento; object storage obrigatório	• Disponível
5k · alta disponibilidade	até 100 RPS / 5.000 usuários	27 VMs com Gitaly Cluster (23 com Gitaly Sharded): mesmos papéis do 3k, com nós maiores de PostgreSQL, Gitaly e GitLab Rails	• Disponível
10k · alta disponibilidade	até 200 RPS / 10.000 usuários	32 VMs com Gitaly Cluster (28 com Gitaly Sharded): papéis do 3k com Redis separado em cache e persistente (3 + 3 VMs) e 4 Sidekiq	○ Sob consulta
25k · alta disponibilidade	até 500 RPS / 25.000 usuários	34 VMs com Gitaly Cluster (30 com Gitaly Sharded): papéis do 10k com 5 GitLab Rails e nós maiores	○ Sob consulta
50k · alta disponibilidade	até 1.000 RPS / 50.000 usuários	41 VMs com Gitaly Cluster (37 com Gitaly Sharded): papéis do 10k com 12 GitLab Rails e nós maiores	○ Sob consulta

- Carga de referência conforme a documentação GitLab: RPS é a métrica principal de dimensionamento; 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. A coluna mostra o RPS de API e a equivalência em usuários.
- Quantidade de VMs e especificação por papel conforme as arquiteturas de referência GitLab (documentação oficial consultada em 29/09/2026). O detalhamento por papel está no documento de pré-requisitos PR-IMP-01.
- As mesmas arquiteturas valem para VMs existentes e para VMs criadas pelo pipeline na AWS ou no Azure. Na infraestrutura criada pelo pipeline, a quantidade de VMs é a da arquitetura, e o tipo de cada papel é o menor tipo da lista do pipeline que atende vCPU e memória da arquitetura (AWS: famílias c7i, m7i e r7i; Azure: Fsv2, Dasv5 e Easv5) ou o tipo definido na configuração.
- Gitaly Sharded usa as mesmas especificações de Gitaly e dispensa os 3 nós Praefect e o PostgreSQL do Praefect. Segundo a documentação GitLab, o Gitaly Cluster (Praefect) oferece tolerância a falhas com complexidade adicional de instalação e gestão.
- Os balanceadores externo e interno são uma VM HAProxy cada, instalada pelo GET; o GET não implementa balanceador em alta disponibilidade em servidores próprios. O balanceador do cliente pode substituir o HAProxy externo, com TLS em passthrough até os nós GitLab Rails (2 ou mais nós); TLS terminado no balanceador do cliente e balanceador interno do cliente são atendidos sob consulta.
- Arquiteturas em estado *Sob consulta*: 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
Infraestrutura criada pelo pipeline (AWS ou Azure)	◦ Integração externa	Depende da conta AWS ou da assinatura Azure do cliente e de uma identidade com permissão para criar os recursos. Módulos Terraform oficiais do GET 3.11.0, sem modificação: na AWS, VMs, rede (nova ou existente), grupos de segurança e buckets; no Azure, VMs e rede virtual em resource group existente (object storage: serviço S3-compatível existente). Estado no GitLab-managed Terraform state do projeto do engajamento, com trava durante cada execução; os endereços internos de cada papel são lidos do estado pela implantação. Registros DNS, certificados e o balanceador do cliente não são criados pelo pipeline. Criação no Google Cloud: sob consulta.
Alta disponibilidade (3k ou mais)	• Nativo	PostgreSQL com Patroni, PgBouncer e Consul, Redis com Sentinel e Gitaly Cluster (Praefect) ou Gitaly Sharded, componentes do Linux package instalados pelo GET conforme a arquitetura de referência.
Balanceador do cliente	◦ Integração externa	Balanceador externo do cliente no lugar do HAProxy do GET, em TLS passthrough: o balanceador encaminha TCP 443 e a porta do SSH do Git aos nós GitLab Rails, e o NGINX de cada nó serve HTTPS com o certificado do cliente (exige certificado do cliente e 2 ou mais nós GitLab Rails). Listeners, health checks e registros DNS são configurados pelo cliente; a verificação pós-implantação testa a página de login pela URL externa. TLS terminado no balanceador e balanceador interno do cliente: sob consulta.
TLS com certificado do cliente	• Requer configuração	Certificado e chave do host externo fornecidos pelo cliente, instalados pelo pipeline na configuração do GET no momento da execução. Opção padrão. Com autoridade certificadora não pública, a cadeia da AC emissora é instalada como confiável em todos os nós com Linux package, que chamam a API interna pela URL externa quando não há balanceador interno.

Recurso	Classificação	Detalhe
TLS com Let's Encrypt	○ Integração externa	Emissão pela autoridade Let's Encrypt (HAProxy externo via acme.sh ou NGINX do Linux package, conforme o GET). Exige e-mail de registro e DNS do host externo resolvível nos nós (erro no preflight quando não resolve).
Object storage S3-compatível	○ Integração externa	Serviço S3-compatível do cliente, na forma consolidada, com um bucket por tipo de objeto (artefatos, dependency proxy, LFS, pacotes, estado Terraform, uploads, Pages, diffs de merge request, CI secure files, backups e, com Container Registry, registry). Obrigatório a partir de 2k; opcional em 1k. Na infraestrutura criada pelo pipeline na AWS, os buckets são criados pelo pipeline (padrão).
GitLab Pages	◄ Requer configuração	URL raiz com DNS wildcard e, com HTTPS, certificado wildcard do domínio do Pages. Em mais de um nó, exige object storage. Com balanceador do cliente em passthrough, o NGINX dos nós GitLab Rails atende o domínio do Pages na porta 443 com o certificado wildcard, e a verificação pós-implantação confere o roteamento do domínio do Pages.
Container Registry	◄ Requer configuração	Container Registry da instância, com storage no object storage S3 (bucket próprio; obrigatório em mais de um nó), em subdomínio (por exemplo registry.) ou em porta própria (nó único ou balanceador do cliente). O certificado TLS precisa cobrir o nome do registry. A verificação pós-implantação não executa push nem pull no registry.
Busca avançada (Advanced Search)	○ Integração externa	Cluster OpenSearch instalado pelo GET em VMs dedicadas (mínimo 1; abaixo de 3 o pipeline emite alerta) ou cluster Elasticsearch/OpenSearch externo do cliente.
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 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.

Recurso	Classificação	Detalhe
LDAP / Active Directory	◦ Integração externa	Servidor LDAP do cliente: servidor, porta (padrão 636), criptografia (simple_tls, start_tls ou plain, esta com alerta), base DN, atributo de login, conta de bind e, opcionalmente, base de grupos e grupo de administradores.
SMTP	◦ Integração externa	Servidor de e-mail do cliente: servidor, porta, TLS (starttls, tls ou nenhum), usuário, domínio, remetente e reply-to.
Backup agendado	• Requer configuração	Agendamento no nó GitLab Rails primário de <code>gitlab-backup create</code> e de cópia da configuração (<code>gitlab-ctl backup-etc</code>), padrão diário às 02:00, retenção padrão de 7 dias; com object storage, os backups vão para o bucket de backups. Em alta disponibilidade, o pipeline emite alerta para revisar a estratégia de backup com o cliente.
Licença	• Requer configuração	Activation code (ativação online) ou arquivo de licença (offline), aplicados pelo GET.
GitLab Geo (dois sites)	• Requer configuração	Dois sites Linux package, primário e secundário, com a mesma versão GitLab, licença GitLab Premium ou GitLab Ultimate no primário e object storage nos dois sites ou em nenhum. A replicação é configurada pelo playbook oficial do GET a partir do site primário, que instala nos nós do secundário a autoridade certificadora da URL do primário. O pipeline não executa failover nem confere a replicação na verificação pós-implantação. GitLab Geo em Kubernetes: sob consulta.
Agente GitLab para Kubernetes	• Requer configuração	Servidor do agente (KAS) habilitado por padrão na instância, publicado pela URL externa; com 2 ou mais nós GitLab Rails, a API privada do KAS entre os nós usa a porta 8155. A verificação pós-implantação confere o serviço do KAS. 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 (fluxo de tokens da versão 16 ou superior da plataforma GitLab), com escopo de instância, grupo ou projeto, tags, proteção, execução sem tag e timeout máximo definidos na configuração, e instalados em VMs (executor Docker), em docker-autoscaler na AWS (Auto Scaling Group existente e dedicado) ou em Kubernetes (chart oficial gitlab-runner). Runner existente é mantido; recriação e remoção entram por merge request. Métricas na porta 9252. Executor shell e docker-autoscaler no Azure: sob consulta.
Atualização de versão	• Nativo	Nova versão definida por merge request, respeitando o caminho oficial de upgrade, com backup sob demanda antes. Atualização com nova execução da implantação, em janela de manutenção. Atualização sem parada na alta disponibilidade (playbook de zero downtime do GET, nó a nó, com backup sob demanda antes): exige nós GitLab Rails dimensionados conforme a arquitetura de referência.
Monitoramento (Prometheus)	• Nativo	Prometheus do Linux package no nó de monitoramento da arquitetura (2k ou mais), instalado pelo GET, com retenção configurável; a verificação pós-implantação confere o Prometheus pronto e com alvos ativos. Grafana gerenciado pela Pointer, com o Prometheus da instância como fonte de dados e login pela instância GitLab: sob consulta. Integração com monitoramento corporativo não faz parte do pipeline.

6. Ferramentas

FERRAMENTA OFICIAL DA GITLAB

GitLab Environment Toolkit (GET)

Imagem oficial do GET 3.11.0, sem modificação. Os playbooks Ansible são executados com inventário estático gerado a partir da configuração do engajamento; object storage, LDAP, SMTP, backup, Container Registry, escopos da busca global e CA da URL externa são aplicados pelos pontos de extensão oficiais (Custom Config e Custom Tasks). Na infraestrutura criada pelo pipeline, os módulos Terraform oficiais do GET para AWS e Azure criam os recursos. O GitLab Geo usa o playbook de Geo do GET, e a atualização sem parada (sob consulta), o playbook de zero downtime.

FERRAMENTA HASHICORP, NA VERSÃO FIXADA PELO GET

Terraform

Somente na infraestrutura criada pelo pipeline: executa os módulos oficiais do GET com o estado no GitLab-managed Terraform state do projeto do engajamento. Plano, aplicação e destruição são etapas separadas; a destruição exige decisão registrada por merge request.

PACOTE OFICIAL DA GITLAB

Linux package

Instalado pelo GET em cada VM a partir do repositório oficial packages.gitlab.com, na versão exata definida no engajamento (GitLab 16.0.0 ou superior; edições Enterprise Edition ou FIPS).

BASE DE AUTOMAÇÃO DO GET

Ansible

Executa os playbooks do GET e os playbooks da Pointer: coleta de fatos dos servidores (preflight, somente leitura), verificação pós-implantação e instalação dos runners da frota em VMs.

FERRAMENTA POINTER

psctl

Valida a configuração do engajamento contra a arquitetura de referência e a matriz de sistemas operacionais, gera o ambiente do GET e o módulo Terraform, lê os endereços do estado do Terraform, lista as credenciais exigidas, avalia o preflight, cria e verifica os runners e os agentes pela API e gera o as-built.

FERRAMENTA OFICIAL DA GITLAB

GitLab Runner

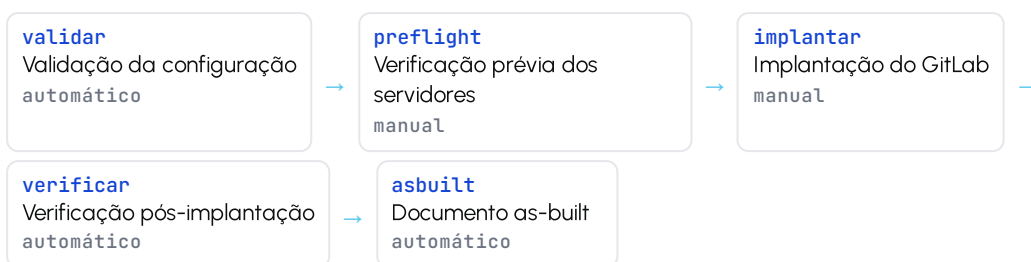
Runner do engajamento: instalado em VM com rota para as VMs, com executor Docker, registrado somente no projeto do engajamento, protegido e sem jobs sem tag; executa preflight, implantação, verificação e rotinas de dia 2, faz somente conexões de saída e é removido no encerramento (ou transferido ao cliente). Frota do cliente: pacote oficial gitlab-runner nas VMs (com o plugin fleeting no docker-autoscaler) ou chart oficial gitlab-runner no Kubernetes, registrado com o token devolvido pela API de criação de runners da instância.

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: Backup sob demanda (**backup-agora**); Atualização de versão sem parada (alta disponibilidade) (**upgrade-zdu**); Verificação sob demanda (**verificar-agora**); Configuração da replicação GitLab Geo (site primário) (**geo-configurar**).

- 1. Criação da infraestrutura (opcional, AWS ou Azure)** — 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 os recursos e registra os endereços internos de cada papel. 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, edição, URL externa, arquitetura, servidores por papel (ou o estado do Terraform) e recursos; gera os arquivos do GET e a lista de credenciais exigidas. Qualquer erro interrompe o pipeline.
- 3. Verificação prévia dos servidores** — no runner do engajamento, somente leitura: conecta por SSH a cada VM e compara sistema operacional, CPU, memória, disco, horário e DNS com a arquitetura de referência. Gera relatório por servidor com erros e alertas.
- 4. Implantação** — com aprovação de pessoa diferente de quem executa: repete a verificação prévia (qualquer erro impede o início) e executa o GET. Uma execução por ambiente de cada vez; tempo máximo de 4 horas.
- 5. Verificação pós-implantação** — automática: estado dos serviços, versão instalada, checagens de integridade da aplicação GitLab e do Gitaly (inclusive a chamada à API interna pela URL externa), cluster Patroni e Praefect quando existem, KAS, roteamento do GitLab Pages, Prometheus com alvos ativos e acesso à página de login pela URL externa.

-
6. **Documento as-built** — gerado a partir da configuração versionada e do resultado da verificação.

Componentes opcionais no mesmo projeto: frota de runners do cliente (validação, aplicação e verificação, por frota) e agentes GitLab para Kubernetes (validação, aplicação e verificação das conexões). Rotinas sob demanda no mesmo pipeline: backup, verificação avulsa, atualização de versão sem parada (alta disponibilidade) e configuração do GitLab Geo a partir do site primário.

8. Metodologia

Assessment → Implantação → Otimização → Transferência de conhecimento.

Fase 1**Assessment**

- Kick-off: escopo, papéis, janelas de manutenção, acessos, aprovadores das execuções em produção e critérios de aceite.
- Discovery de implantação: usuários atuais e projeção de 1 a 3 anos, RPS medido quando já existe instância GitLab, uso de monorepos, CI/CD e pacotes, sistema operacional, rede, DNS, certificados, object storage, LDAP, SMTP, backup (RTO e RPO) e licenciamento, com registro de cada resposta no registro de decisões.
- Definição da forma da infraestrutura: VMs existentes ou criadas pelo pipeline (conta ou assinatura, região, rede, identidade, políticas da organização, cotas e tags exigidas).
- Definição dos recursos opcionais: balanceador do cliente, Container Registry, segundo site GitLab Geo, escopos da busca global, frota de runners (modalidades, escopos e tags) e clusters a conectar pelo agente GitLab para Kubernetes.
- Escolha da arquitetura de referência e, em

Fase 2**Implantação**

- Infraestrutura criada pelo pipeline: plano revisado com o cliente e aplicação aprovada no ambiente protegido.
- Instalação do runner do engajamento na VM definida pelo cliente, ambiente protegido com aprovação e cadastro das credenciais.
- Verificação prévia contra os servidores reais; 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.
- GitLab Geo: implantação dos dois sites e configuração da replicação a partir do site primário.
- Frota de runners e agentes GitLab para Kubernetes: criação na instância, instalação e verificação pelos componentes do pipeline.

Fase 3**Otimização**

- Execução do backup agendado ao menos uma vez e, quando necessário, backup sob demanda.
- Validação com o cliente do login LDAP e do e-mail de teste, quando habilitados.
- Ajustes de configuração por merge request e nova execução do pipeline, inclusive tags, proteção e recriação de runners da frota.
- 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: troca da senha inicial do root e cópia do gitlab-secrets.json para local seguro fora do servidor.
- Sessões sobre a arquitetura implantada, o pipeline e as rotinas de dia 2 (backup, verificação avulsa, frota de runners e atualização de versão pelo caminho oficial de upgrade).
- Entrega do repositório do engajamento, com opção de espelhamento na instância GitLab do cliente.
- Infraestrutura criada 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, credenciais revogadas no projeto do engajamento, projeto arquivado, termo de

alta disponibilidade, do modo do Gitaly. - Envio do documento de pré-requisitos PR-IMP-01 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.			aceite e relatório de encerramento.
--	--	--	--

9. Entregáveis

Entregável	Descrição	Formato
Configuração do engajamento	Configuração do ambiente (arquitetura, servidores por papel ou infraestrutura a criar, 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 a infraestrutura: 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	Tabela por servidor (papel, sistema operacional, vCPU, memória, espaço livre em /var/opt/gitlab, resultado) com erros e alertas, e fatos coletados dos servidores.	Markdown e JSON (artefatos do pipeline)
Relatório de verificação pós-implantação	Resultado de cada checagem (OK ou FALHOU): estado dos serviços, versão, checagens de integridade da aplicação GitLab e do Gitaly, Patroni e Praefect quando existem, KAS, roteamento do GitLab Pages, Prometheus e resposta HTTP 200 da página de login pela URL externa.	JSON (artefato do pipeline)

Entregável	Descrição	Formato
As-built	Cliente, contrato, responsável, pipeline e commit, versões GitLab e do GET, arquitetura de referência, servidores por papel (com provedor, região, tipos e tags na infraestrutura criada pelo pipeline), recursos habilitados (inclusive balanceador do cliente, Container Registry, GitLab Geo, escopos da busca global e Prometheus), buckets exigidos, 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 backup agendado (e do backup sob demanda, quando executado) no ambiente entregue.	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, login LDAP e e-mail de teste validados quando habilitados, backup agendado executado ao menos uma vez, runners da frota online e agentes conectados quando contratados) e registro do encerramento.	Documento

10. Premissas

- VMs existentes, provisionadas pelo cliente conforme a arquitetura escolhida e o documento de pré-requisitos, ou VMs criadas pelo pipeline na conta AWS ou Azure do cliente; nos dois casos, sistema operacional recém-instalado (o GET exige instalação limpa do sistema operacional).
- Infraestrutura criada pelo pipeline: identidade de nuvem do cliente com permissão para criar os recursos da arquitetura (na conta AWS ou no resource group do Azure), sem políticas da organização que neguem essas ações, e cotas da região suficientes. Registros DNS, certificados e balanceador do cliente continuam com o cliente.
- Assinatura GitLab Premium ou GitLab Ultimate para o go-live; a edição Community Edition não é aceita pelo pipeline.

- Instalação online: as VMs acessam `packages.gitlab.com` e os repositórios do sistema operacional; o runner acessa `gitlab.com` e `registry.gitlab.com` por HTTPS, diretamente ou por proxy.
- Um ambiente (por exemplo homologação ou produção) por arquitetura de referência; cada ambiente adicional, inclusive o segundo site GitLab Geo, é uma inclusão própria no projeto do engajamento.
- Com balanceador do cliente, listeners, health checks e registros DNS são configurados pelo cliente antes da implantação.
- Frota de runners: token de API do cliente com permissão para criar runners no escopo escolhido; no docker-autoscaler, Auto Scaling Group existente e dedicado, com imagem que já tenha o Docker Engine ou comando de prontidão definido.
- 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 entram somente por merge request.
- Os objetos do object storage não entram no backup da plataforma GitLab (documentação GitLab); a cópia desses dados usa o backup do próprio provedor de object storage.
- Tempos e cronograma do engajamento definidos na proposta.

Fora do escopo

- Criação de infraestrutura no Google Cloud pelo pipeline: sob consulta (VMs existentes no Google Cloud entram como infraestrutura existente).
- Balanceador do cliente com TLS terminado no balanceador e balanceador interno do cliente no lugar do HAProxy interno: sob consulta.
- Configuração do balanceador do cliente (listeners, health checks e certificado), registros DNS e emissão de certificados: responsabilidade do cliente, inclusive na infraestrutura criada pelo pipeline.
- Atualização de versão sem parada com nós GitLab Rails abaixo do dimensionamento da arquitetura de referência: atualização com janela de manutenção.
- Failover do GitLab Geo: não automatizado pelo pipeline nesta versão; a replicação não faz parte da verificação pós-implantação.
- 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.
- Instalação sem acesso à internet (*air-gapped*, pacote offline): roadmap.
- Autenticação SAML ou OIDC: não é configurada pelo pipeline nesta versão.

- Gestão de servidor DNS e integração com monitoramento corporativo (fora do escopo do GET).
- Migração de dados de outra plataforma para a instância implantada (ofertas de migração PS-MIG-01 a PS-MIG-04).

Responsabilidades do cliente

- Provisionar as VMs por papel, com sistema operacional suportado, volumes montados, NTP e DNS, ou, com infraestrutura criada pelo pipeline, fornecer a conta ou assinatura, a identidade com as permissões e as cotas da região.
- Criar o usuário de implantação com sudo sem senha em todas as VMs existentes e entregar a chave SSH (na infraestrutura criada pelo pipeline, a chave pública é gravada nas VMs pelo pipeline).
- Fornecer certificado TLS (ou optar por Let's Encrypt), registros DNS, object storage com os buckets criados (salvo buckets criados pelo pipeline na AWS) e licença GitLab Premium ou Ultimate.
- Configurar o balanceador do cliente, quando usado no lugar do HAProxy do GET.
- Disponibilizar a VM do runner e liberar as regras de rede: saída HTTPS, SSH do runner até as VMs e portas entre os nós.
- Emitir os tokens de API da frota de runners e dos agentes e o acesso aos clusters, quando contratados.
- Designar patrocinador, ponto focal técnico, aprovador das execuções e janela de manutenção.
- Validar login LDAP, e-mail de teste e demais critérios de aceite.
- Detalhamento no documento de pré-requisitos PR-IMP-01.

Detalhamento no documento de pré-requisitos PR-IMP-01, enviado após a escolha do serviço.

11. Duração

A duração depende da arquitetura de referência e do número de VMs, da forma da infraestrutura (existente ou criada pelo pipeline), dos recursos opcionais contratados (GitLab Geo, frota de runners, agentes), da prontidão dos pré-requisitos e das janelas de mudança. Cada execução da etapa de implantação pelo pipeline tem tempo máximo de 4 horas, e cada aplicação da infraestrutura, de 2 horas. O prazo do engajamento é definido na proposta.

12. Documentos relacionados

- PS-00 · Catálogo de Serviços
- PR-IMP-01 · Pré-requisitos: Implantação da plataforma GitLab em servidores Linux

-
- **PS-IMP-02** · Implantação da plataforma GitLab em Kubernetes
 - **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-01 · 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.