DEVSECOPS COURSE

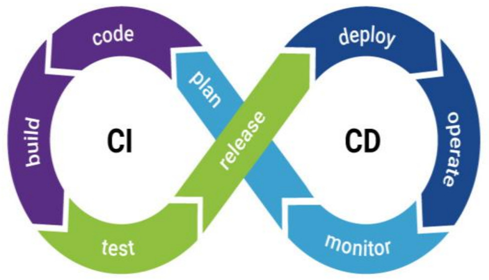
CONTINOUS INTEGRATION CONTINOUS DEPLOYMENT

TRAINER: TRAN HUU HOA

AGENDA

INTRODUCTION

- Continuous integration (CI) focuses on blending the software work products of individual developers together into a repository. This can be done several times a day, with the primary purpose being to enable early detection of integration bugs while also allowing for tighter cohesion and more development collaboration.
- The aim of continuous delivery (CD) is to minimize the friction points that are inherent in the deployment or release processes. Typically, a team's implementation involves automating each of the steps for build deployments so that a safe code release can be done at any moment in time.
- Continuous deployment (CD) is a higher degree of automation, in which a build/deployment occurs automatically whenever a major change is made to the code.



INTRODUCTION

WHY CICD

- Reduction of delivery risk
- To encode the process, we need to know the process
- Better visibility on change
- Open up more avenues for review and increased audit compliance
- Increase efficiency and delivery options
- Enhanced learning from failure

CICD PIPELINES DESIGN

For continuous integration:

- Decoupled stages: each step in CI should do a single focused task
- Repeatable: automated in a way that consistently repeatable
- Fail fast: fail the first sign of trouble

For continuous delivery/deployment:

- Design with the system in mind that cover as many parts of a deployment as possible (application | infrastructure | configuration | Data)
- Pipelines: continually increase confidence as you move towards production
- Globally unique version: know the state of the system at any time and be able to demonstrate diffirent between current and future state

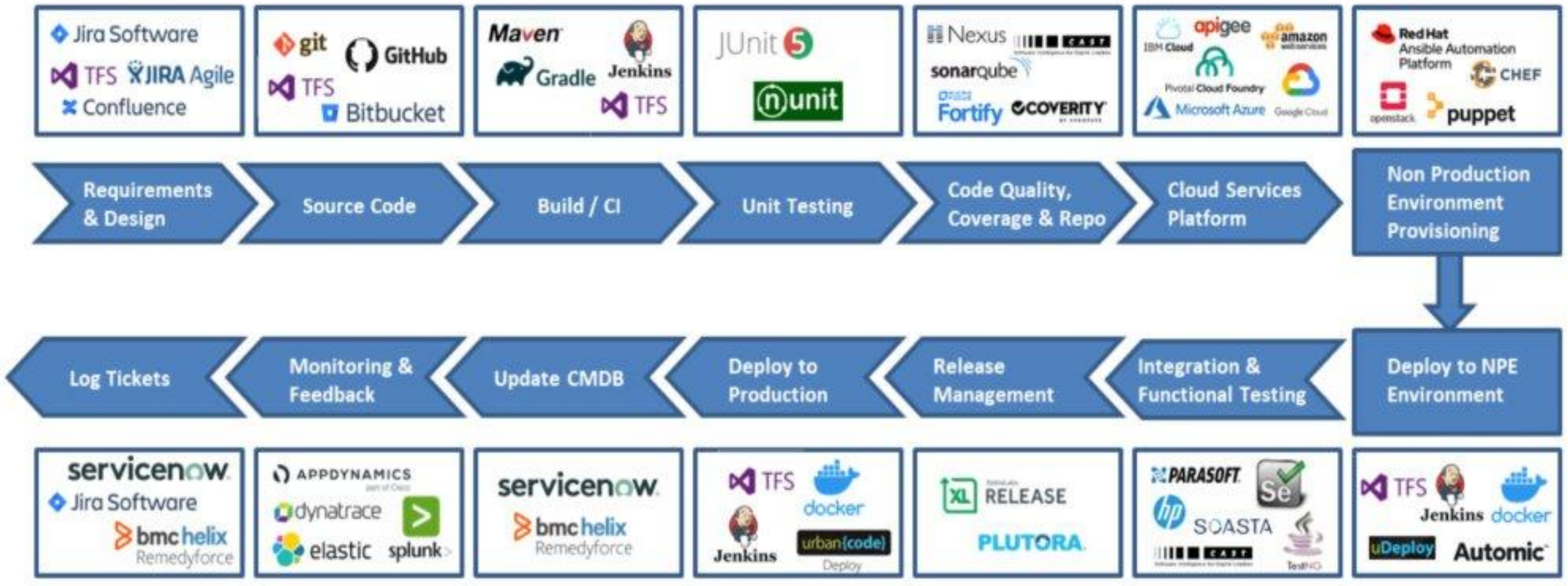
CICD PIPELINES DESIGN

Best practices

- Centralized artifacts
- Everything as code
- 1 seperated pipeline of CICD for 1 microservice only
- Deployment more frequently
- Reduce lead time (time from plan to production)
- CI with continuous test (both automation and performance)
- CI with continuous security test
- Adopting a gitflow model is mandatory

CICD PIPELINES DESIGN

Sample toolchains



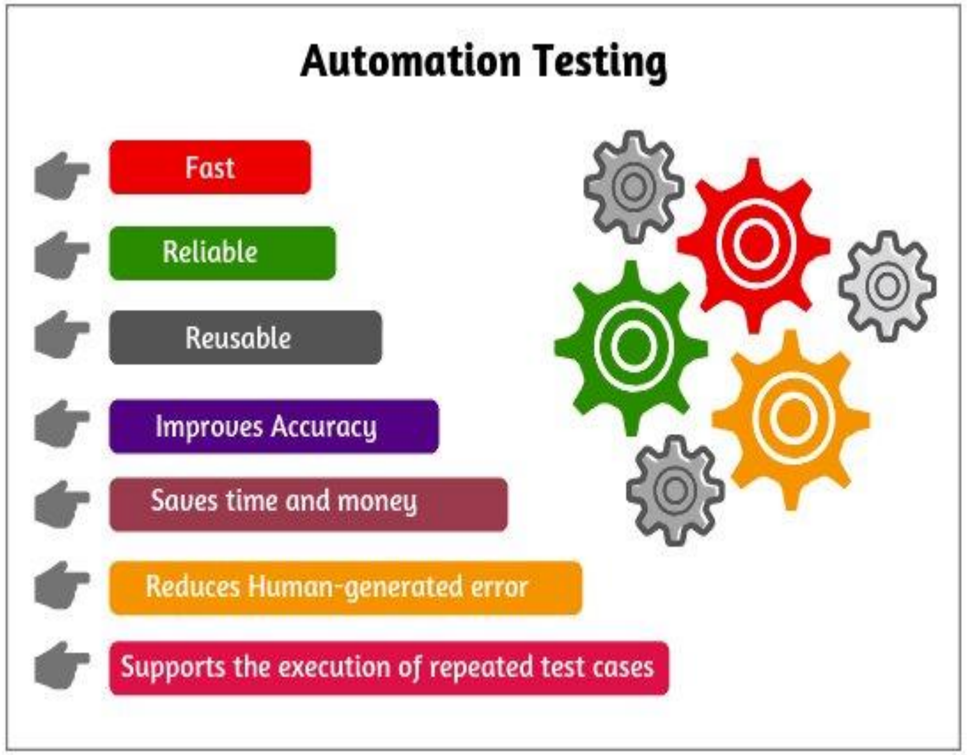
CICD PIPELINES DESIGN

CI implementation rules

- Maintain a single source code repository for one service, one purpose
- Automate the build
- Make your build self testing
- Keep the build fast
- Everyone can see what is happening
- Build template, SOE images as much as possible

CICD PIPELINES DESIGN

Integration with Automation test

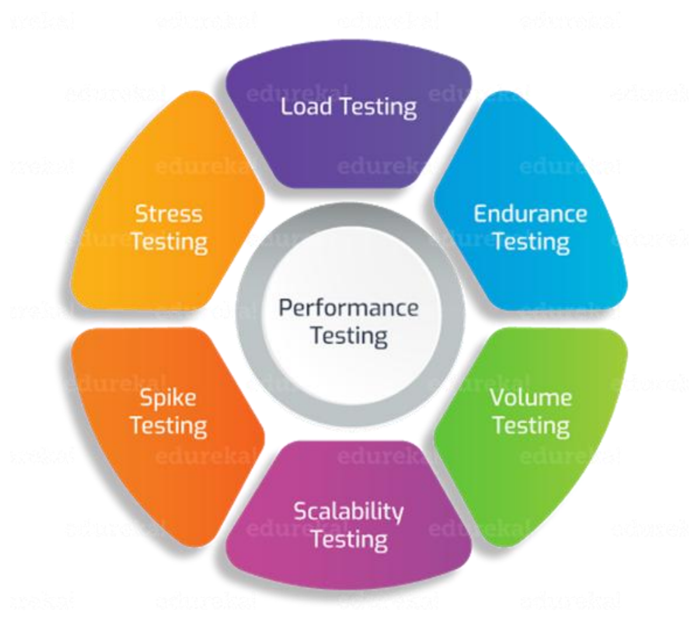


Automation Testing

- Fast
- Reliable
- Reusable
- Improves Accuracy
- Saves time and money
- Reduces Human-generated error
- Supports the execution of repeated test cases

CICD PIPELINES DESIGN

Integration with performance test



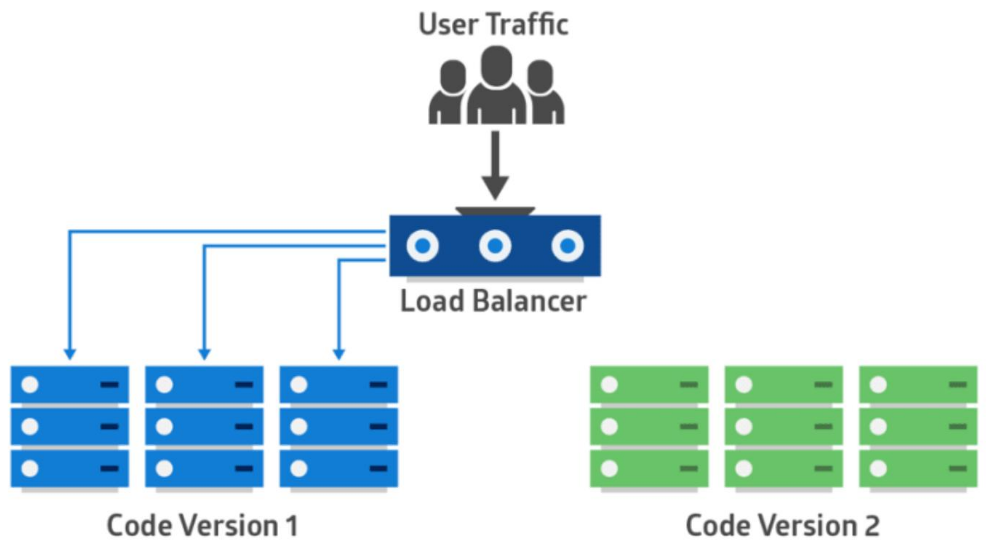
CICD PIPELINES DESIGN

Deployment patterns:

- Blue-Green pattern
- Canary pattern
- Rolling update pattern

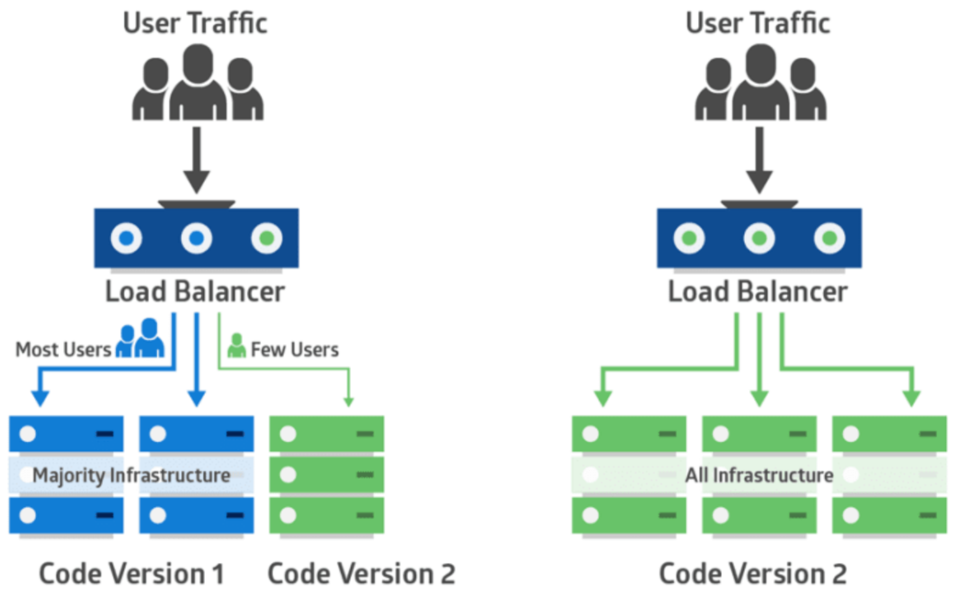
CICD PIPELINES DESIGN

Blue-Green pattern



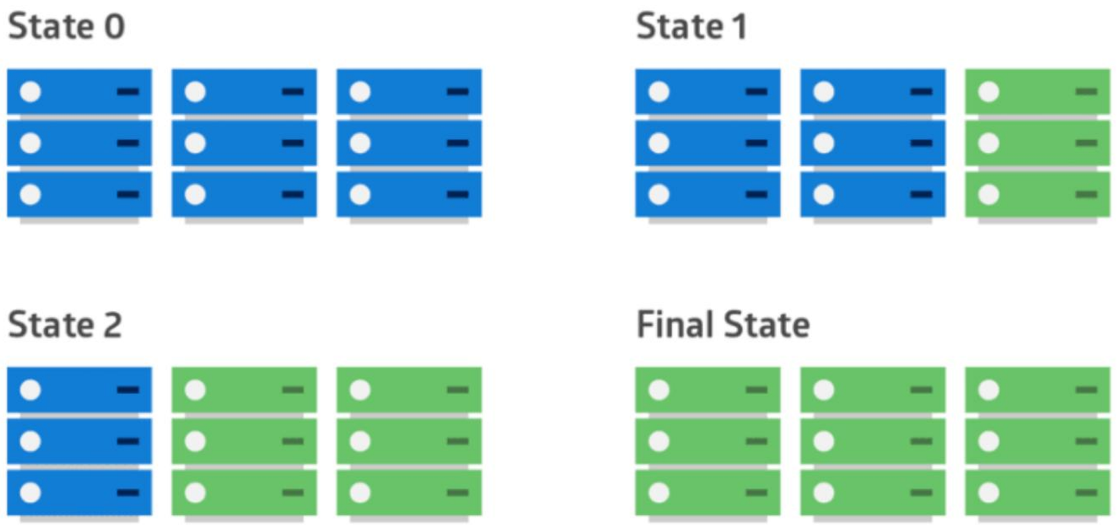
CICD PIPELINES DESIGN

Canary pattern



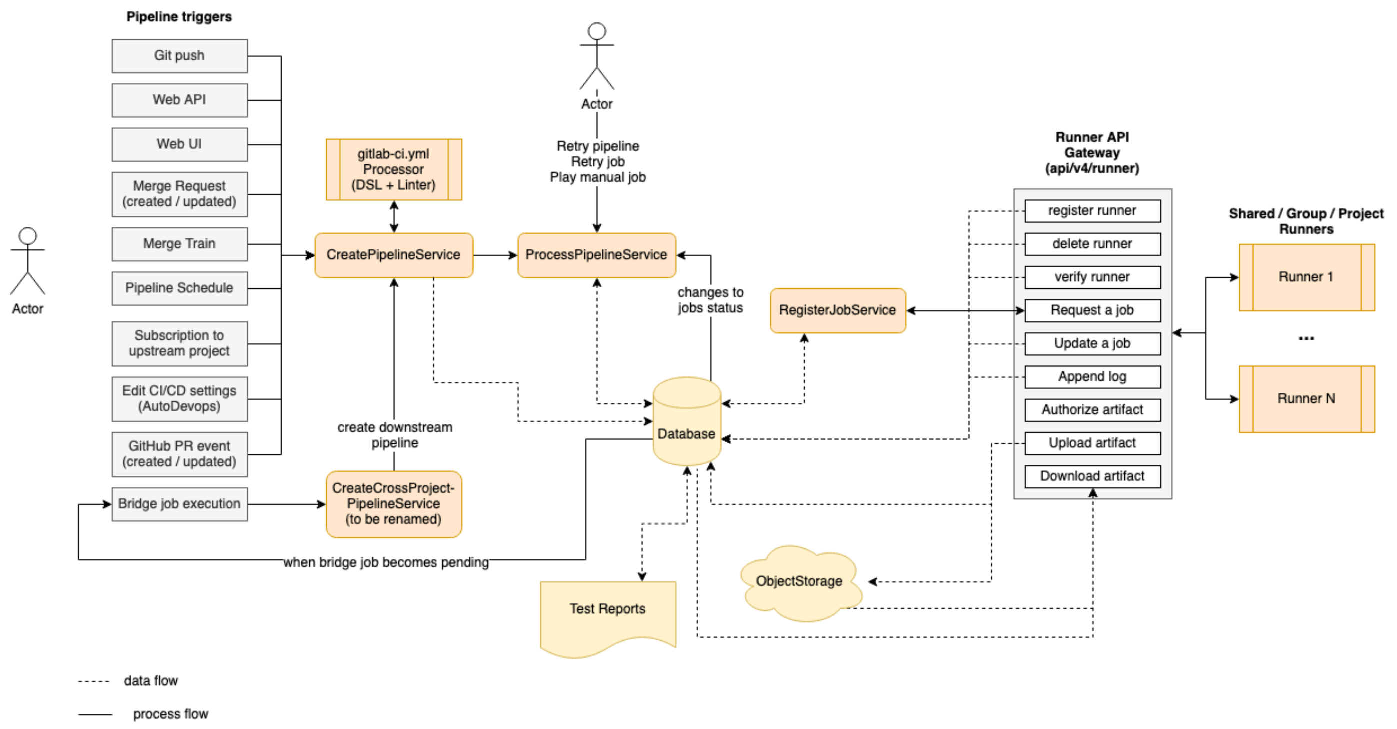
CICD PIPELINES DESIGN

Rolling update pattern



GITLAB RUNNER

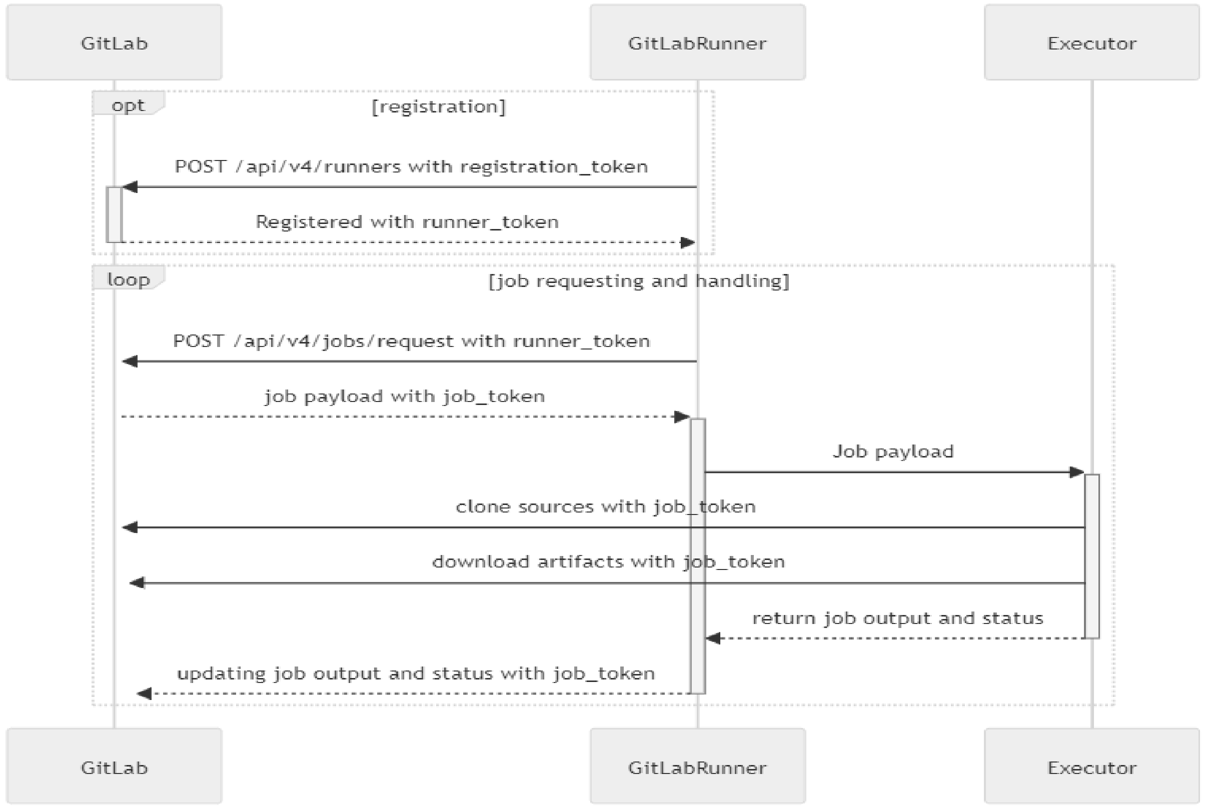
Architecture



GITLAB RUNNER

GitLab Runner is an application that works with GitLab CI/CD to run jobs in a pipeline.

- SaaS runners
- self-managed runners


GitLab
GitLabRunner
Executor
opt
[registration]
POST /api/v4/runners with registration_token
Registered with runner_token
loop
[job requesting and handling]
POST /api/v4/jobs/request with runner_token
job payload with job_token
Job payload
clone sources with job_token
download artifacts with job_token
return job output and status
updating job output and status with job_token
GitLab
GitLabRunner
Executor

GITLAB RUNNER

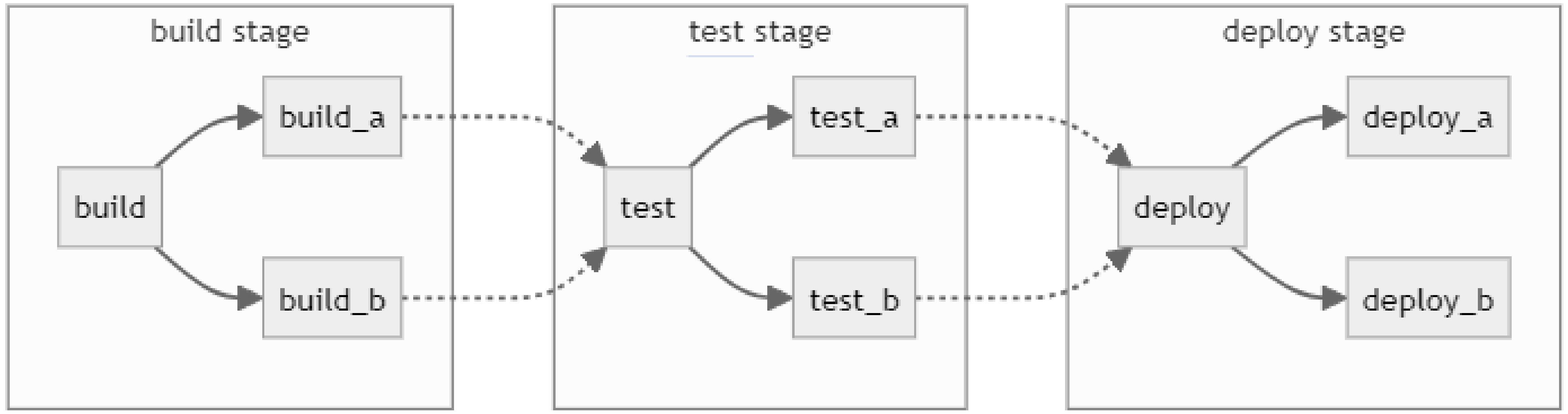
- Run multiple jobs concurrently.
- Use multiple tokens with multiple servers (even per-project).
- Limit the number of concurrent jobs per-token.
- Jobs can be run:
  - ✓ Locally.
  - ✓ Using Docker containers.
  - ✓ Using Docker containers and executing job over SSH.
  - ✓ Using Docker containers with autoscaling on different clouds and virtualization hypervisors.
  - ✓ Connecting to a remote SSH server.

GITLAB RUNNER

- Is written in Go and distributed as single binary without any other requirements.
- Supports Bash, PowerShell Core, and Windows PowerShell.
- Works on GNU/Linux, macOS, and Windows (pretty much anywhere you can run Docker).
- Allows customization of the job running environment.
- Automatic configuration reload without restart.
- Easy to use setup with support for Docker, Docker-SSH, Parallels, or SSH running environments.
- Enables caching of Docker containers.
- Easy installation as a service for GNU/Linux, macOS, and Windows.
- Embedded Prometheus metrics HTTP server.
- Referee workers to monitor and pass Prometheus metrics and other job-specific data to GitLab.

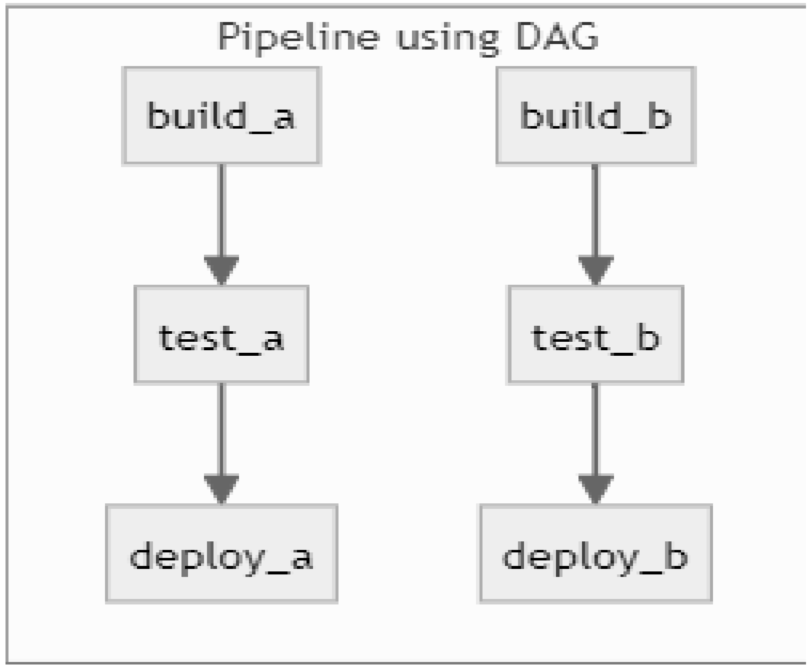
GITLAB RUNNER

Basic pipelines



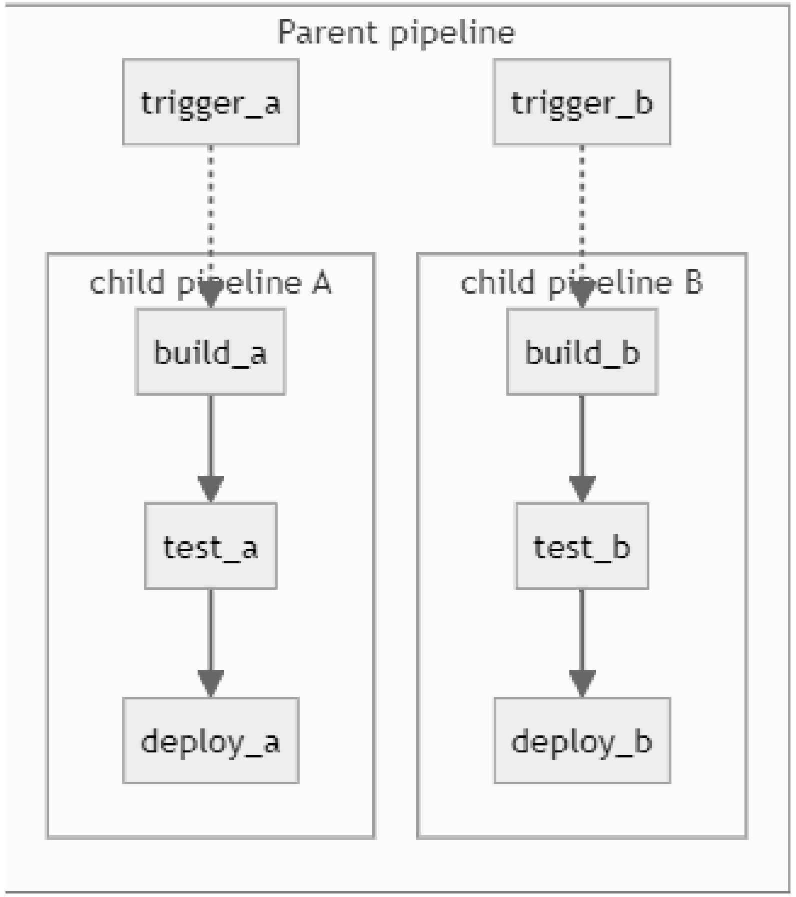
GITLAB RUNNER

Directed Acyclic Graph Pipelines



GITLAB RUNNER

Parent-child pipelines



JENKINS

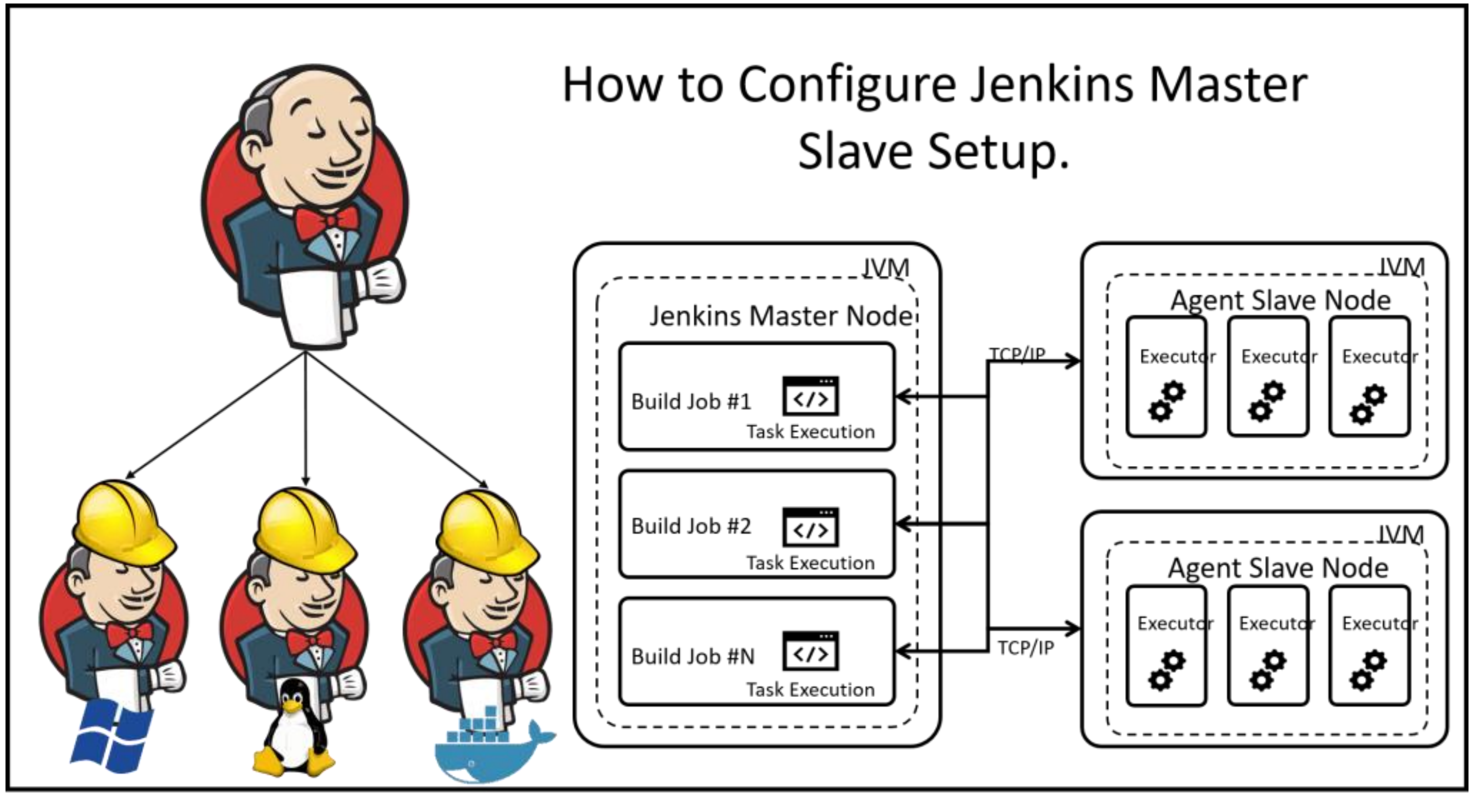
Concept: The leading open-source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

Why Jenkins:

- Support both Continuous Integration and Continuous Delivery
- Easy installation and configuration
- Plugins and Extensible
- Distributed

JENKINS

Architecture



JENKINS

- Store Pipeline definitions in a SCM
- When writing a pipeline definition, use Declarative syntax (Groovy)
- Use shared libraries but avoid large ones
- Only use Scripted syntax when it doesn't make sense to use Declarative plus a shared library
- Don't use input within an agent
- Wrap your input in a timeout
- Do all work within an agent
- Acquire agents within parallel steps
- Avoid script security exceptions
- Avoid complex code
- Reduce repetition of similar Pipeline steps
- Avoid large global variable declaration files
- Keep always secure
- Always Backup The “JENKINS_HOME” Directory

ARGOCD

ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes

ArgoCD can pull updated code from Git repositories and deploy it directly to Kubernetes resources. It enables developers to manage both infrastructure configuration and application updates in one system.

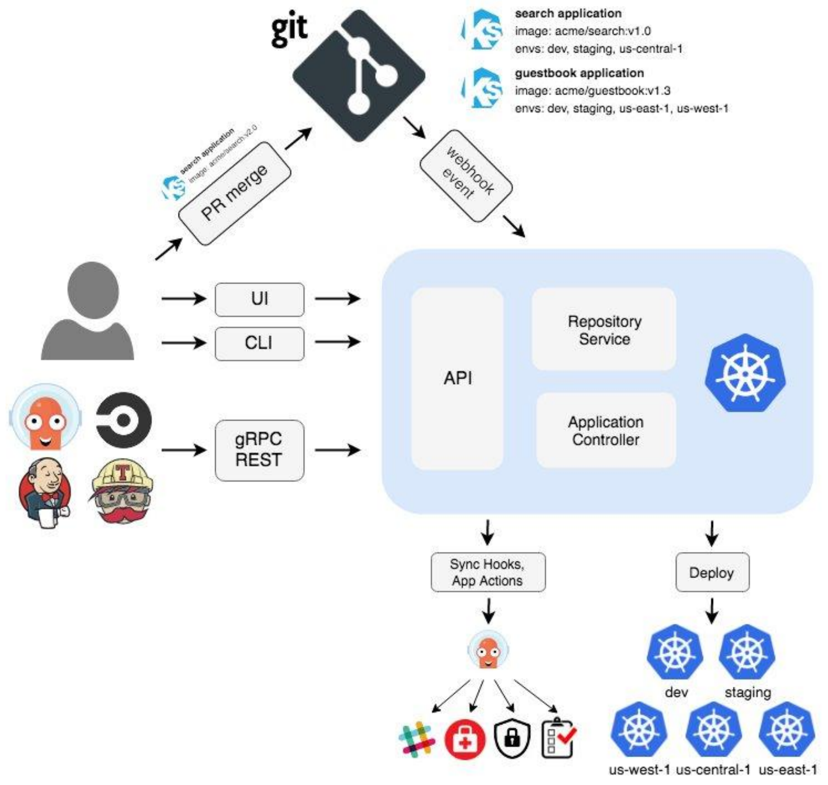
Application definitions, configurations, and environments should be declarative and version controlled. Application deployment and lifecycle management should be automated, auditable, and easy to understand

ARGOCD

A sample flow:

1. A developer makes changes to an application, pushing a new version of Kubernetes resource definitions to a Git repo.
2. Continuous integration is triggered, resulting in a new container image saved to a registry.
3. A developer issues a pull request, changing Kubernetes manifests, which are created either manually or automatically.
4. The pull request is reviewed and changes are merged to the main branch. This triggers a webhook which tells Argo CD a change was made.
5. Argo CD clones the repo and compares the application state with the current state of the Kubernetes cluster. It applies the required changes to cluster configuration.
6. Kubernetes uses its controllers to reconcile the changes required to cluster resources, until it achieves the desired configuration.
7. Argo CD monitors progress and when the Kubernetes cluster is ready, reports that the application is in sync.
8. ArgoCD also works in the other direction, monitoring changes in the Kubernetes cluster and discarding them if they don't match the current configuration in Git.

ARGOCD


git
search application
image: acme/search:v1.0
envs: dev, staging, us-central-1
guestbook application
image: acme/guestbook:v1.3
envs: dev, staging, us-east-1, us-west-1
search application
image: acme/search:v2.0
PR merge
webhook event
UI
CLI
gRPC REST
API
Repository Service
Application Controller
Sync Hooks, App Actions
Deploy
dev
staging
us-west-1
us-central-1
us-east-1

ARGOCD

- Automated deployment of applications to specified target environments
- Support for multiple config management/templating tools (Kustomize, Helm, Jsonnet, plain-YAML)
- Ability to manage and deploy to multiple clusters
- SSO Integration (OIDC, OAuth2, LDAP, SAML 2.0, GitHub, GitLab, Microsoft, LinkedIn)
- Multi-tenancy and RBAC policies for authorization
- Rollback/Roll-anywhere to any application configuration committed in Git repository
- Health status analysis of application resources
- Automated configuration drift detection and visualization
- Automated or manual syncing of applications to its desired state
- Web UI which provides real-time view of application activity
- CLI for automation and CI integration
- Webhook integration (GitHub, BitBucket, GitLab)
- Access tokens for automation
- PreSync, Sync, PostSync hooks to support complex application rollouts (e.g.blue/green & canary upgrades)
- Audit trails for application events and API calls
- Prometheus metrics
- Parameter overrides for overriding helm parameters in Git

ARGOCD

Methods

Installation manifests

Kustomize

Helm

ARGOCD

<u>GitOps based to make sure:</u>

Ensure that the clusters have similar states for configuration, monitoring, or storage.
Recover or recreate clusters from a known state.
Create clusters with a known state.
Apply or revert configuration changes to multiple clusters.
Associate template configuration with different environments.

ARGOCD

- Separating Source Code and Configuration Repositories
- Selecting a Suitable Number of Deployment Configuration Repositories
- Testing Manifests Before Each Commit
- Preventing External Changes to Manifests