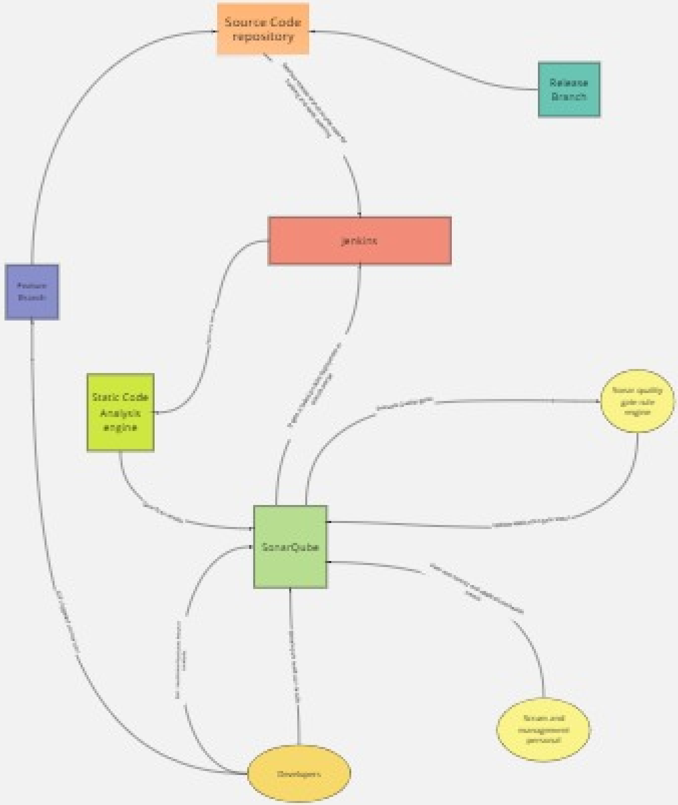
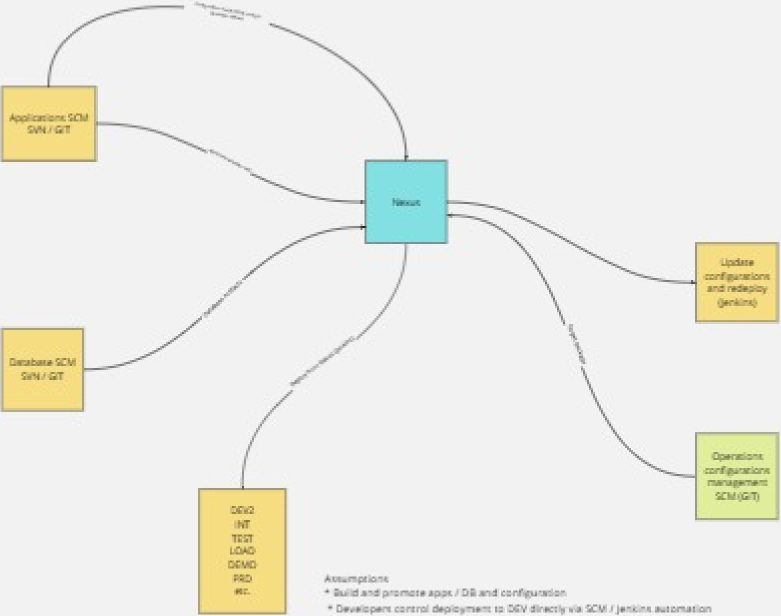
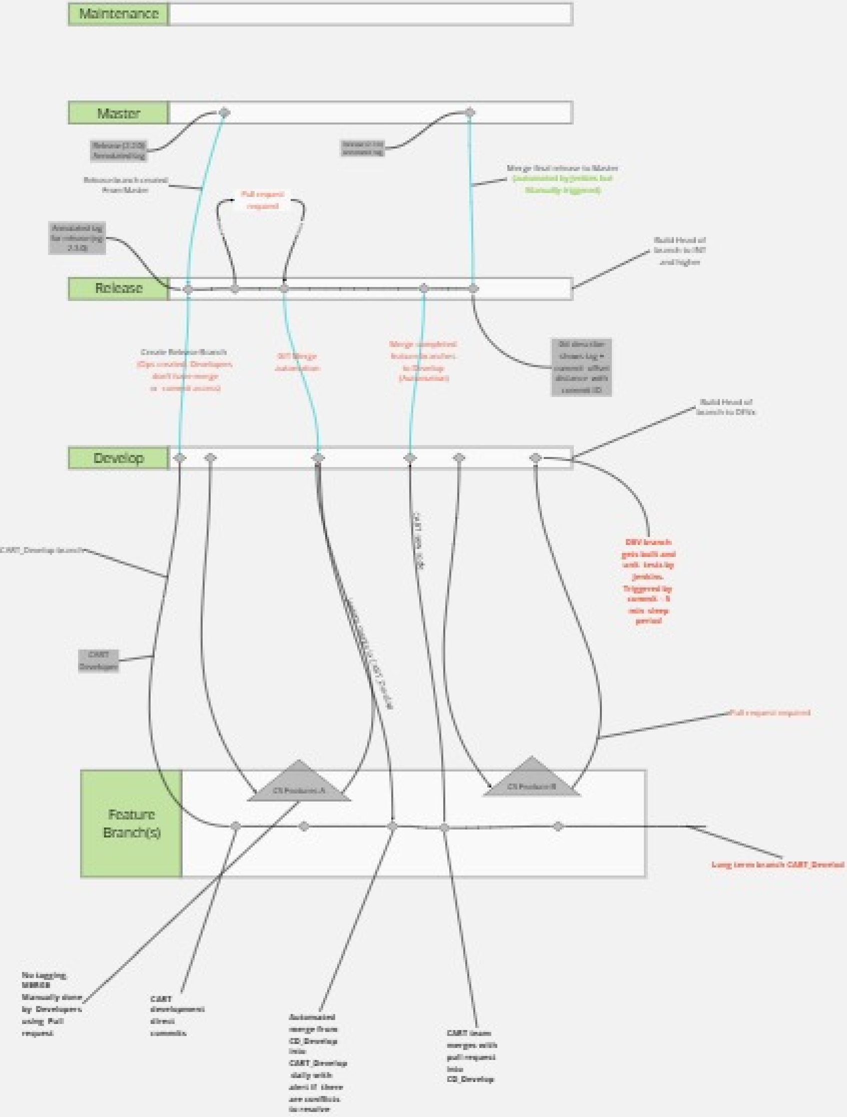
Source code scanning detail flow

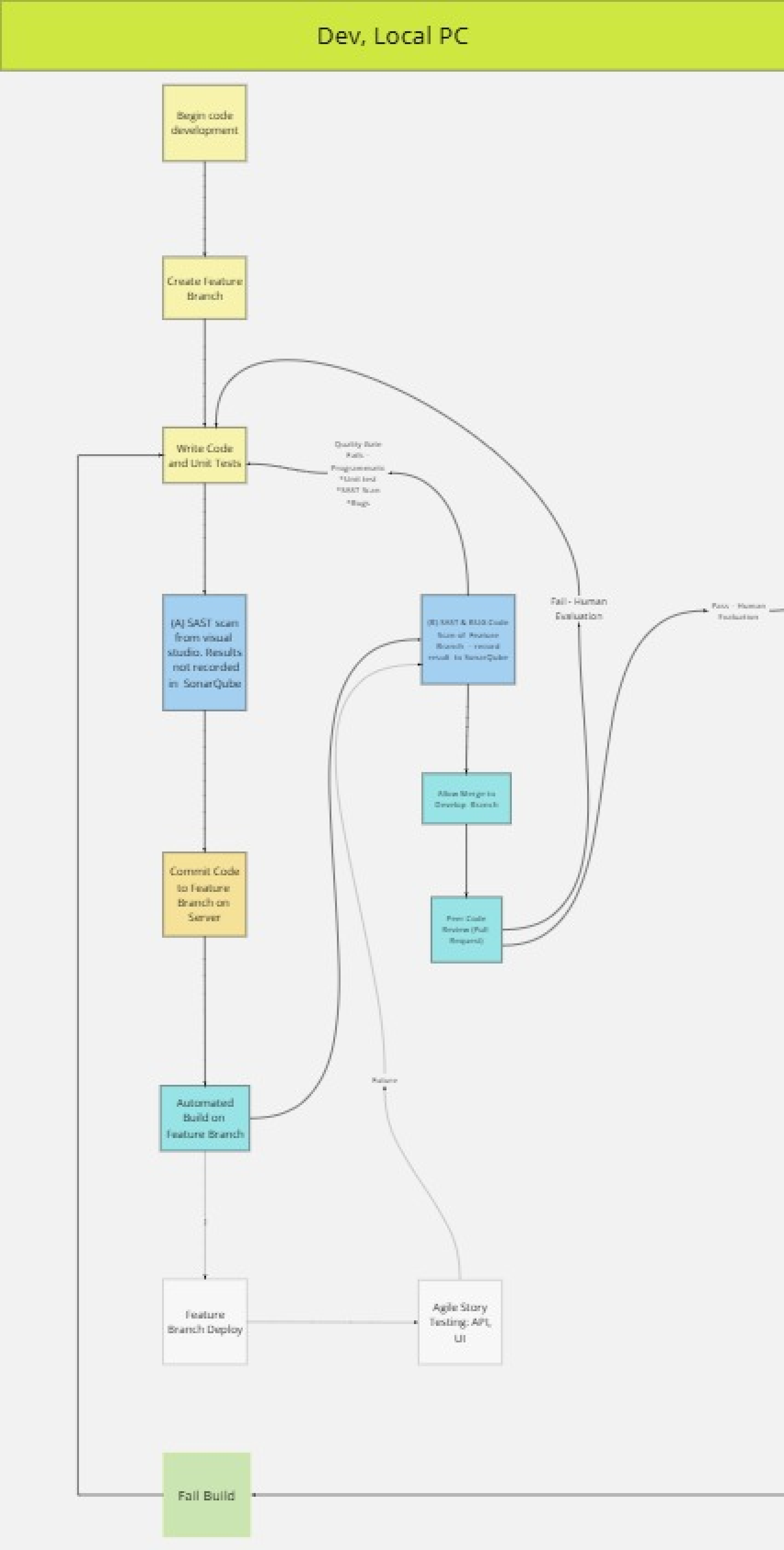


Source Code repository
Release Branch
Jenkins
Feature Branch
Static Code Analysis engine
Sonar quality gate rule engine
SonarQube
Scrum and management personal
Developers


Applications SCM
SVN / GIT
Nexus
Update configurations and redeploy (Jenkins)
Database SCM
SVN / GIT
DEV2
INT
TEST
LOAD
DEMO
PRD
etc.
Operations configurations management SCM (GIT)

Assumptions
* Build and promote apps / DB and configuration
 * Developers control deployment to DEV directly via SCM / Jenkins automation
* There is a need to change config after developer driven packages are created
 * Need to manage config updates, versions, new package deployment out of band
 * Need to understand version dependency of config to application, DB to application and application to application


Maintenance
Master
Release (2.3.0)
Annotated tag
Release 2.3.1
annotated tag
Release branch created
from Master
Merge final release to Master
(automated by Jenkins but
Manually triggered)
Pull request
required
Annotated tag
for release (eg
2.3.0)
Release
Build Head of
branch to INT
and higher
Git describe
shows tag +
commit offset
distance with
commit ID
Create Release Branch
(Ops created, Developers
don't have merge
or commit access)
Git Merge
Automation
Merge completed
feature branches
to Develop
(Automation)
Build Head of
branch to DEVs
Develop
CART_Develop branch
DEV branch
gets built and
unit tests by
Jenkins.
Triggered by
commit - 5
min sleep
period
CART
Developer
Pull request required
CS Feature A
CS Feature B
Feature
Branch(s)
Long term branch CART_Develop
No tagging
MERGE
Manually done
by Developers
using Pull
request
CART
development
direct
commits
Automated
merge from
CD_Develop
into
CART_Develop
daily with
alert if there
are conflicts
to resolve
CART team
merges with
pull request
into
CD_Develop



Dev, Local PC

Begin code development

Create Feature Branch

Write Code and Unit Tests

Quality Gate Fails - Programmatic *Unit test *SAST Scan *Bugs

Fail - Human Evaluation

(A) SAST scan from visual studio. Results not recorded in SonarQube

(B) SAST & BUG Code Scan of Feature Branch - record result to SonarQube

Pass - Human Evaluation

Allow Merge to Develop Branch

Commit Code to Feature Branch on Server

Peer Code Review (Pull Request)

Failure

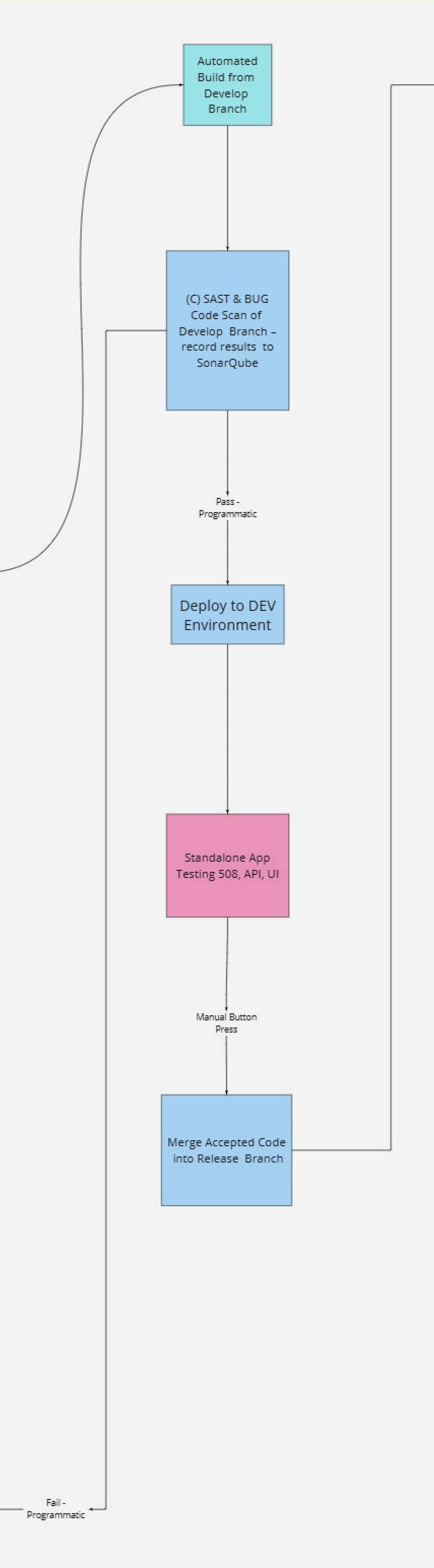
Automated Build on Feature Branch

Feature Branch Deploy

Agile Story Testing: API, UI

Fail Build

Dev VLAN



Automated Build from Develop Branch

(C) SAST & BUG Code Scan of Develop Branch – record results to SonarQube

Pass - Programmatic

Deploy to DEV Environment

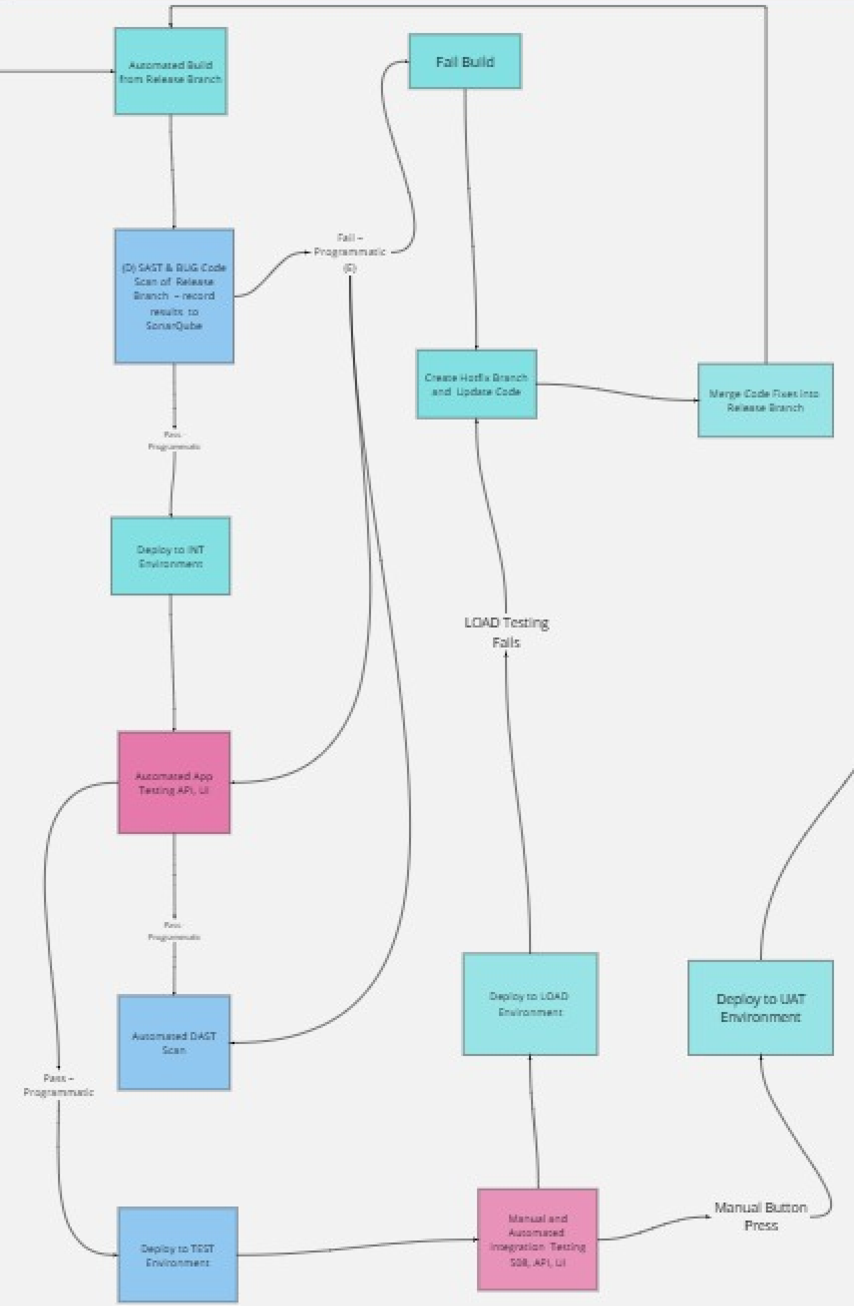
Standalone App Testing 508, API, UI

Manual Button Press

Merge Accepted Code into Release Branch

Fail - Programmatic

Pre-Production VLAN



Automated Build from Release Branch

Fail Build

(D) SAST & BUG Code Scan of Release Branch – record results to SonarQube

Fail – Programmatic (E)

Create Hotfix Branch and Update Code

Merge Code Fixes Into Release Branch

Pass - Programmatic

Deploy to INT Environment

LOAD Testing Fails

Automated App Testing API, UI

Pass - Programmatic

Automated DAST Scan

Deploy to LOAD Environment

Deploy to UAT Environment

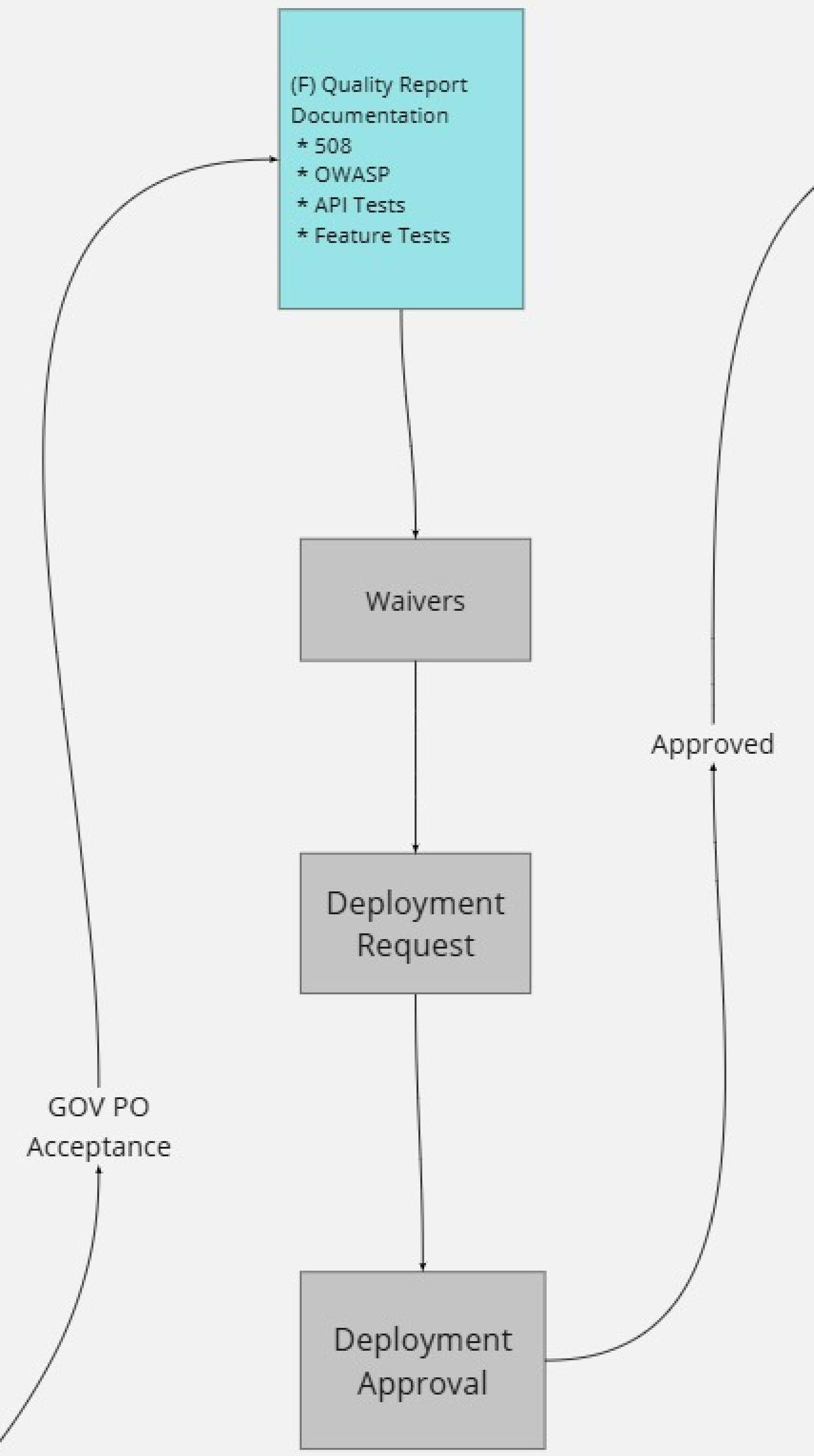
Pass – Programmatic

Deploy to TEST Environment

Manual and Automated Integration Testing SDB, API, UI

Manual Button Press

DRT



(F) Quality Report Documentation
- 508
- OWASP
- API Tests
- Feature Tests

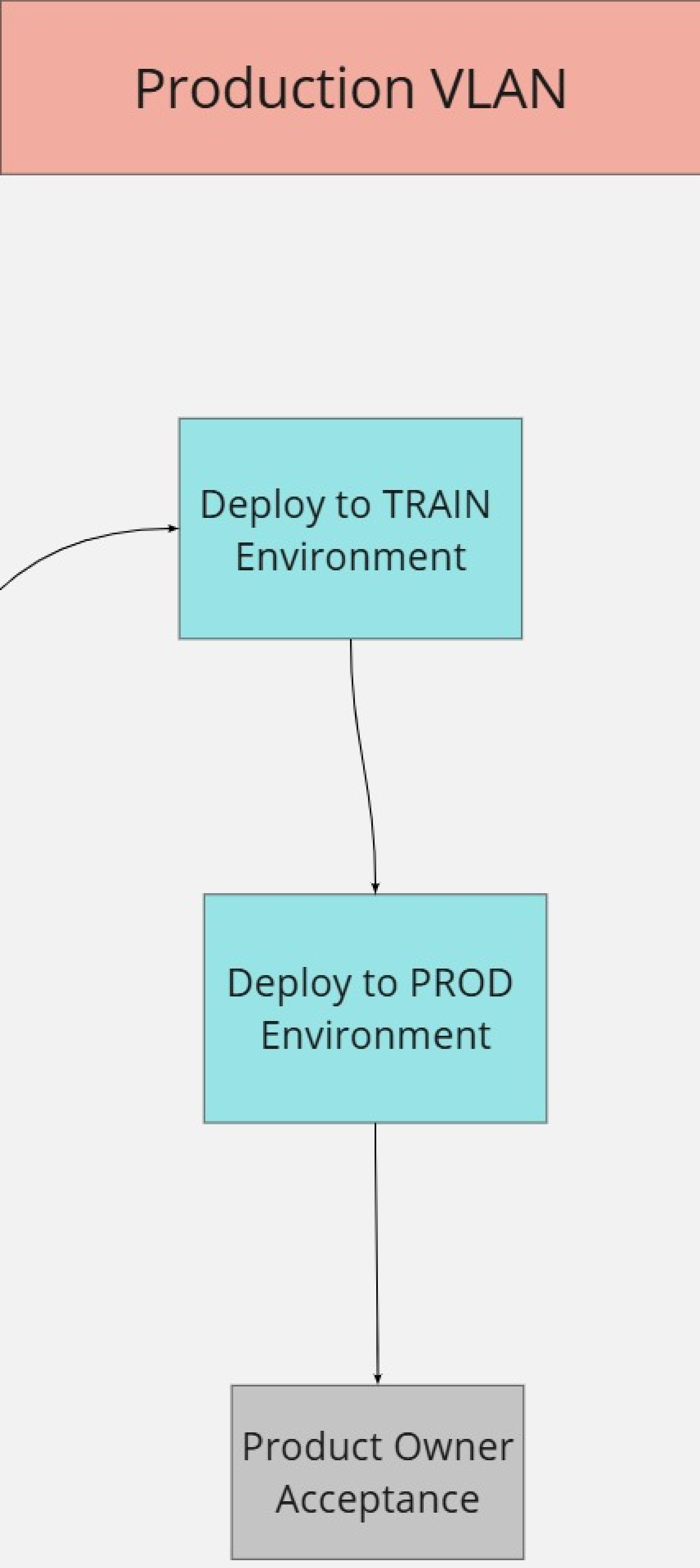
Waivers

Deployment Request

Deployment Approval

Approved

GOV PO Acceptance

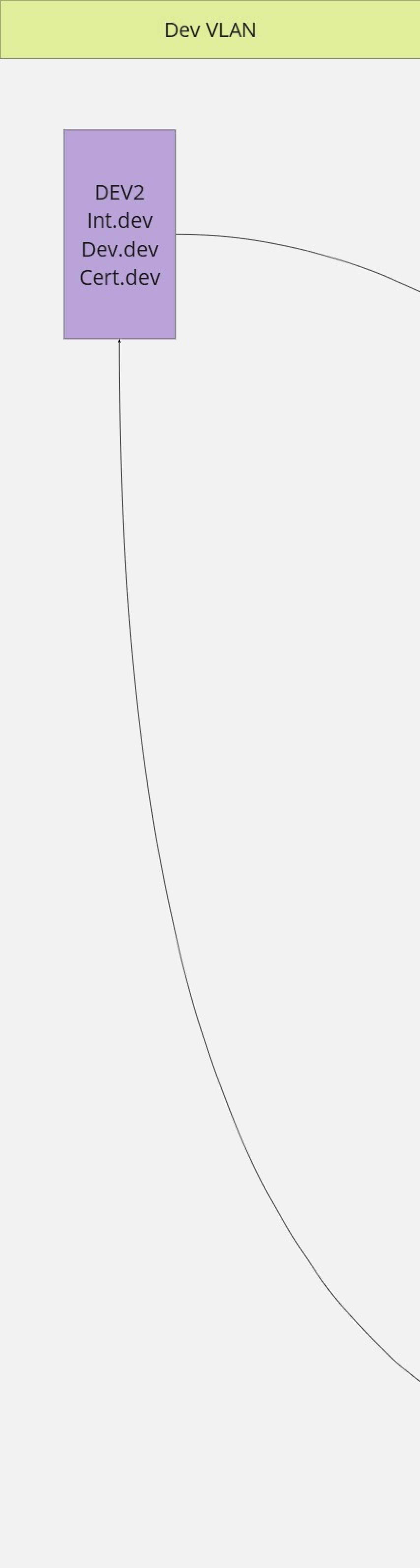


Production VLAN

Deploy to TRAIN Environment

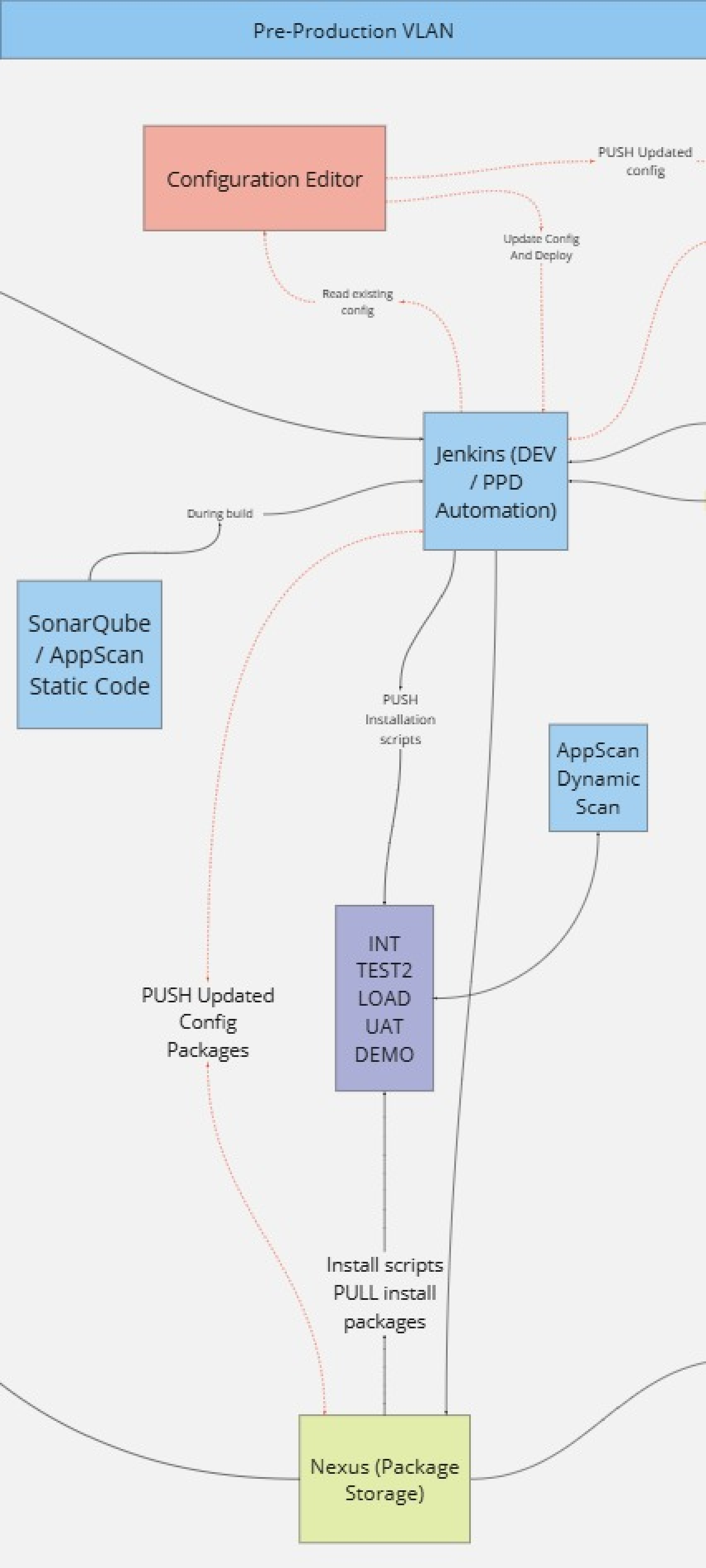
Deploy to PROD Environment

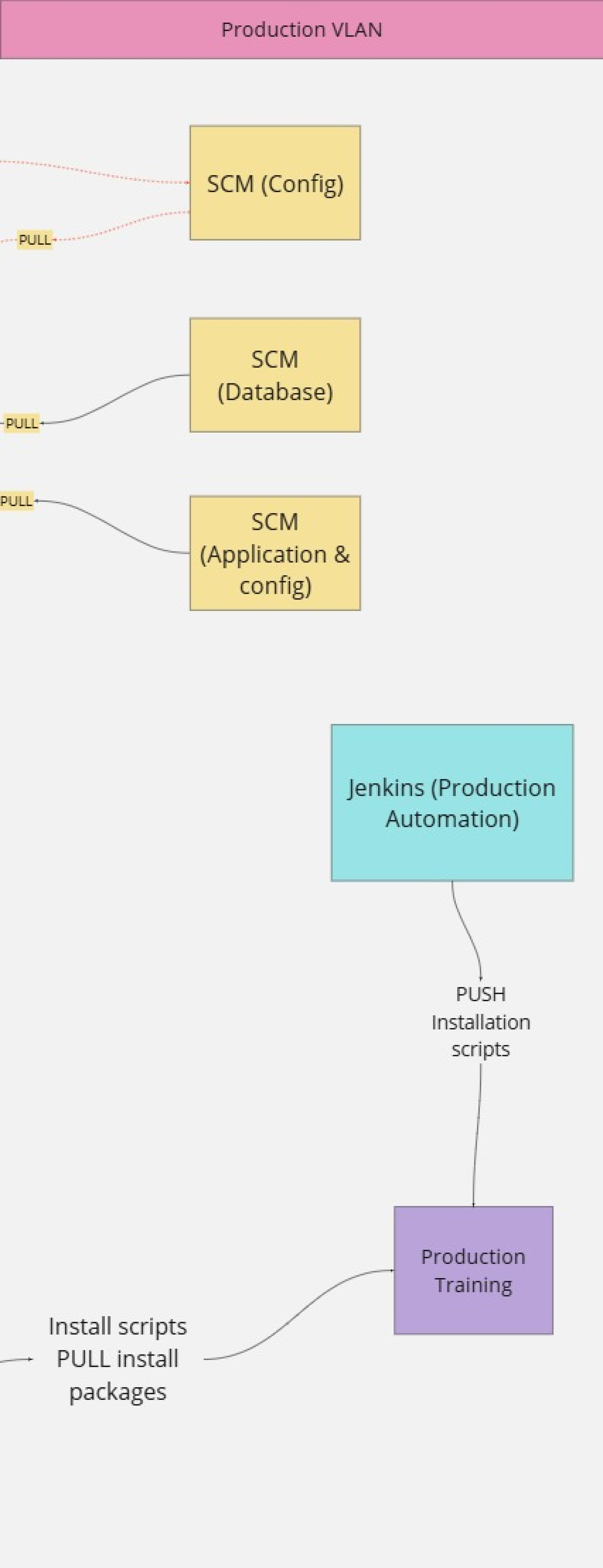
Product Owner Acceptance

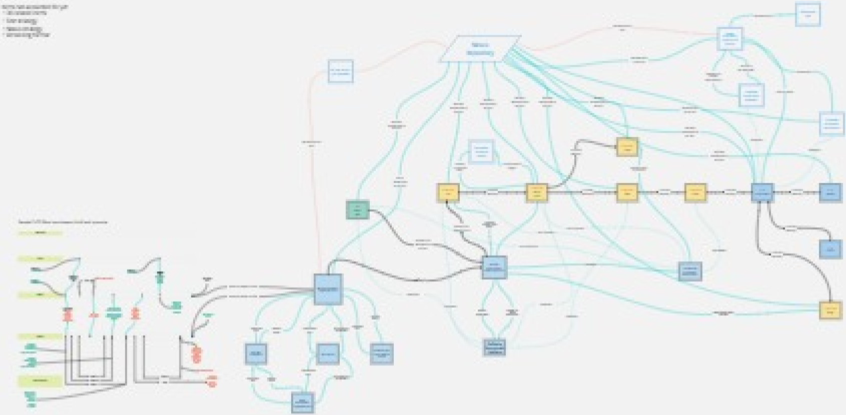


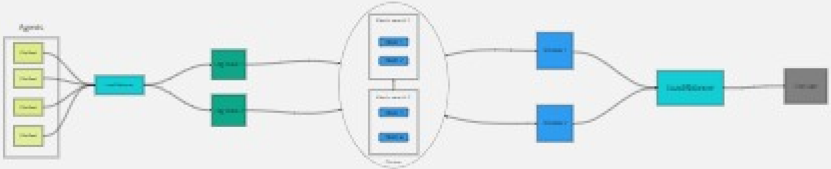
Dev VLAN

DEV2
Int.dev
Dev.dev
Cert.dev


Pre-Production VLAN
Configuration Editor
PUSH Updated config
Update Config And Deploy
Read existing config
Jenkins (DEV / PPD Automation)
During build
SonarQube / AppScan Static Code
PUSH Installation scripts
AppScan Dynamic Scan
INT
TEST2
LOAD
UAT
DEMO
PUSH Updated Config Packages
Install scripts PULL install packages
Nexus (Package Storage)


Production VLAN
SCM (Config)
PULL
SCM (Database)
PULL
PULL
SCM (Application & config)
Jenkins (Production Automation)
PUSH Installation scripts
Production Training
Install scripts PULL install packages

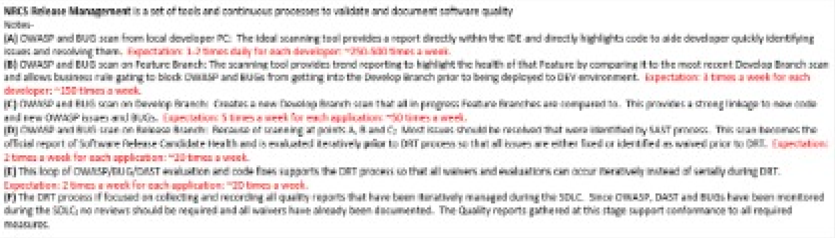



Agents
LoadBalancer

CI/CD Release flow

The CI/CD service has a general set of capabilities that are in a desired order. Some are linear and some can be performed in parallel. Not all applications are intended to have the identical set of environments so long as the capabilities are available for each application.

- Red items in diagram are partially implemented and are planned.
- Hashed boxes ad dotted lines are planned future capabilities
- Not all applications or teams have adopted the full scope of flow represented



NRCS Release Management is a set of tools and continuous processes to validate and document software quality

Notes-

[A] OWASP and BUG scan from local developer PC. The ideal scanning tool provides a report directly within the IDE and directly highlights code to aide developer quickly identifying issues and resolving them. Expectation: 1-2 times daily for each developer: ~250-500 times a week.

[B] OWASP and BUG scan on Feature Branch: The scanning tool provides trend reporting to highlight the health of that Feature by comparing it to the most recent Develop Branch scan and allows business rule gating to block OWASP and BUGs from getting into the Develop Branch prior to being deployed to DEV environment. Expectation: 3 times a week for each developer: ~150 times a week.

[C] OWASP and BUG scan on Develop Branch: Creates a new Develop Branch scan that all in progress Feature Branches are compared to. This provides a strong linkage to new code and new OWASP issues and BUGs. Expectation: 5 times a week for each application: ~50 times a week.

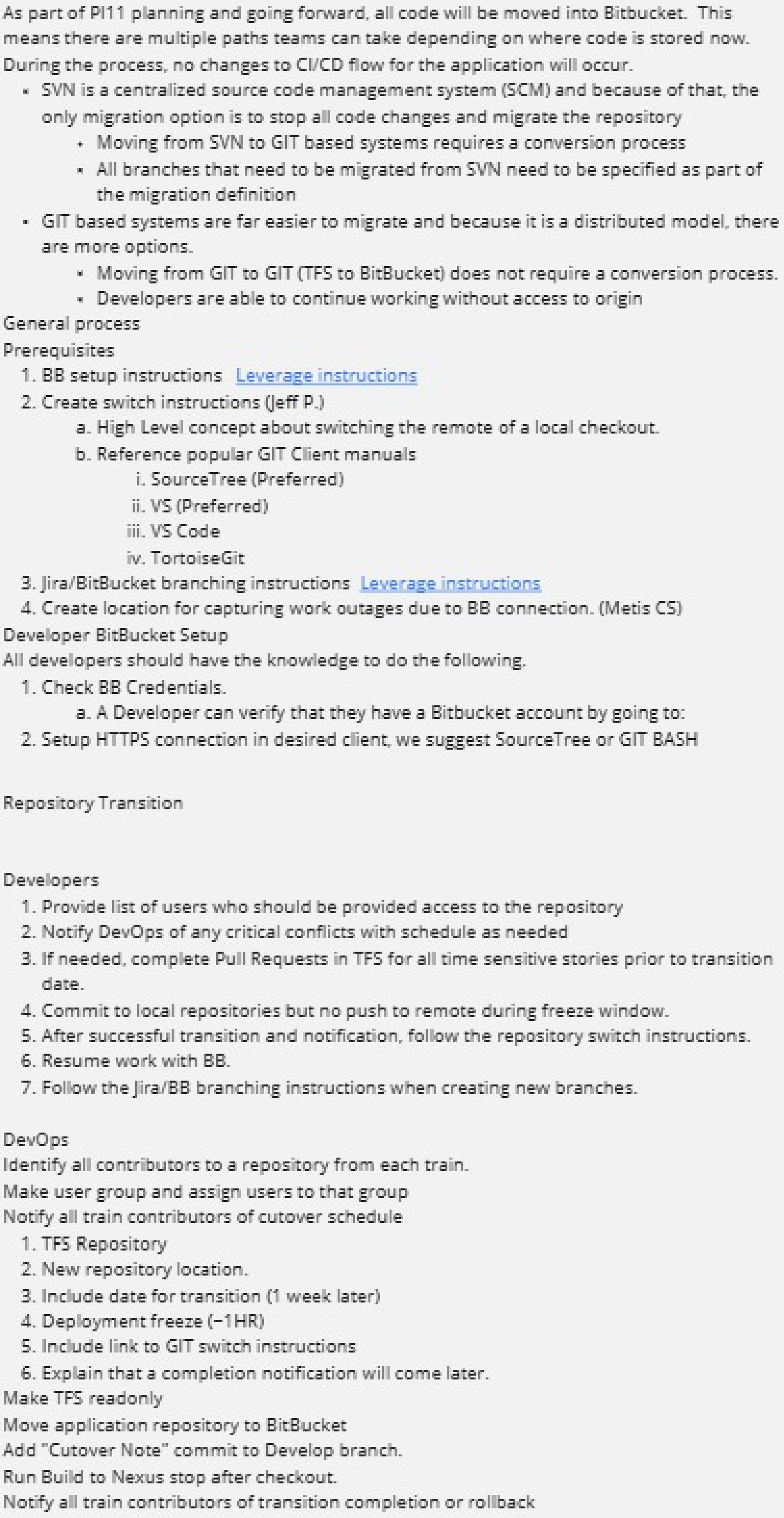
[D] OWASP and BUG scan on Release Branch: Because of scanning at points A, B and C; Most issues should be resolved that were identified by SAST process. This scan becomes the official report of Software Release Candidate Health and is evaluated iteratively prior to DRT process so that all issues are either fixed or identified as waived prior to DRT. Expectation: 2 times a week for each application: ~20 times a week.

[E] This loop of OWASP/BUG/DAST evaluation and code fixes supports the DRT process so that all waivers and evaluations can occur iteratively instead of serially during DRT. Expectation: 2 times a week for each application: ~20 times a week.

[F] The DRT process if focused on collecting and recording all quality reports that have been iteratively managed during the SDLC. Since OWASP, DAST and BUGs have been monitored during the SDLC; no reviews should be required and all waivers have already been documented. The Quality reports gathered at this stage support conformance to all required measures.

Source code, build artifacts and deployment of build artifacts

VLANs in DISC data center and logical distribution of CI/CD systems and environments



As part of PI11 planning and going forward, all code will be moved into Bitbucket. This means there are multiple paths teams can take depending on where code is stored now.

During the process, no changes to CI/CD flow for the application will occur.

- SVN is a centralized source code management system (SCM) and because of that, the only migration option is to stop all code changes and migrate the repository
    - Moving from SVN to GIT based systems requires a conversion process
    - All branches that need to be migrated from SVN need to be specified as part of the migration definition
- GIT based systems are far easier to migrate and because it is a distributed model, there are more options.
    - Moving from GIT to GIT (TFS to BitBucket) does not require a conversion process.
    - Developers are able to continue working without access to origin

General process

Prerequisites

1. BB setup instructions   Leverage instructions
2. Create switch instructions (Jeff P.)
    a. High Level concept about switching the remote of a local checkout.
    b. Reference popular GIT Client manuals
        i. SourceTree (Preferred)
        ii. VS (Preferred)
        iii. VS Code
        iv. TortoiseGit
3. Jira/BitBucket branching instructions Leverage instructions
4. Create location for capturing work outages due to BB connection. (Metis CS)

Developer BitBucket Setup

All developers should have the knowledge to do the following.

1. Check BB Credentials.
    a. A Developer can verify that they have a Bitbucket account by going to:
2. Setup HTTPS connection in desired client, we suggest SourceTree or GIT BASH

Repository Transition

Developers

1. Provide list of users who should be provided access to the repository
2. Notify DevOps of any critical conflicts with schedule as needed
3. If needed, complete Pull Requests in TFS for all time sensitive stories prior to transition date.
4. Commit to local repositories but no push to remote during freeze window.
5. After successful transition and notification, follow the repository switch instructions.
6. Resume work with BB.
7. Follow the Jira/BB branching instructions when creating new branches.

DevOps

Identify all contributors to a repository from each train.

Make user group and assign users to that group

Notify all train contributors of cutover schedule

1. TFS Repository
2. New repository location.
3. Include date for transition (1 week later)
4. Deployment freeze (~1HR)
5. Include link to GIT switch instructions
6. Explain that a completion notification will come later.

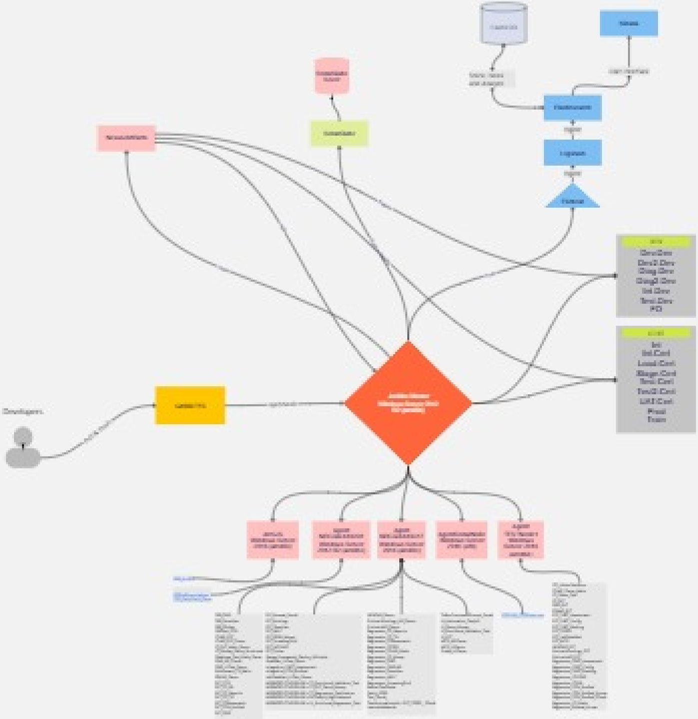
Make TFS readonly

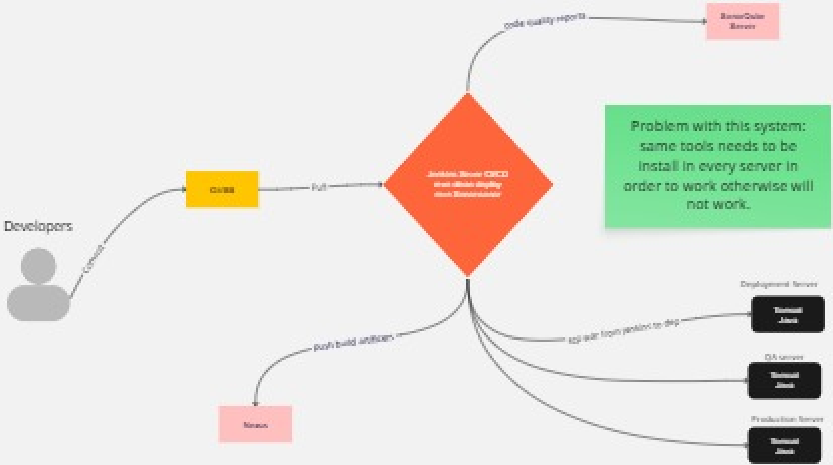
Move application repository to BitBucket

Add "Cutover Note" commit to Develop branch.

Run Build to Nexus stop after checkout.

Notify all train contributors of transition completion or rollback

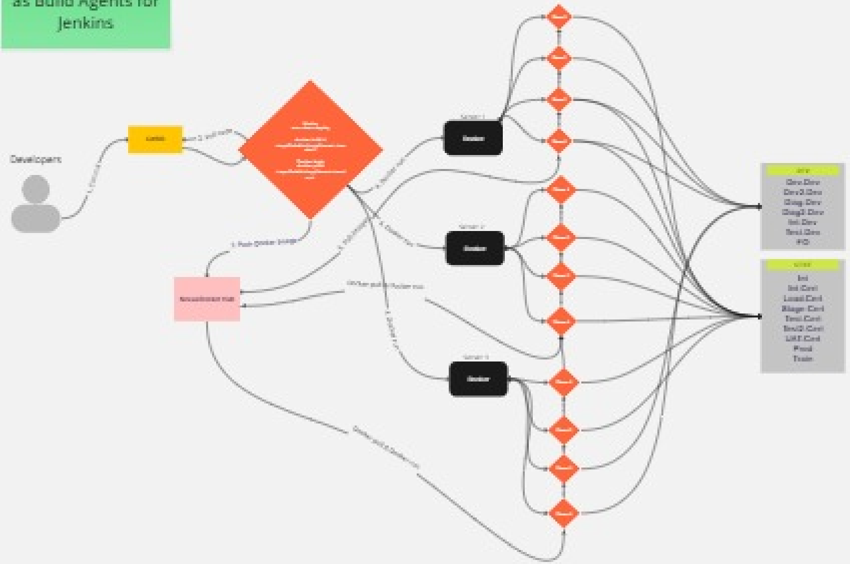

Developers
Dev.Dev
Dev2.Dev
Diag.Dev
Diag2.Dev
Int.Dev
Test.Dev
PD
Int
Int.Cert
Load.Cert
Stage.Cert
Test.Cert
Test2.Cert
UAT.Cert
Prod
Train

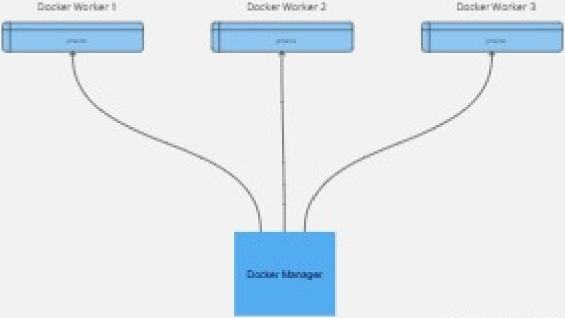


Problem with this system: same tools needs to be install in every server in order to work otherwise will not work.

Docker Containers as Build Agents for Jenkins

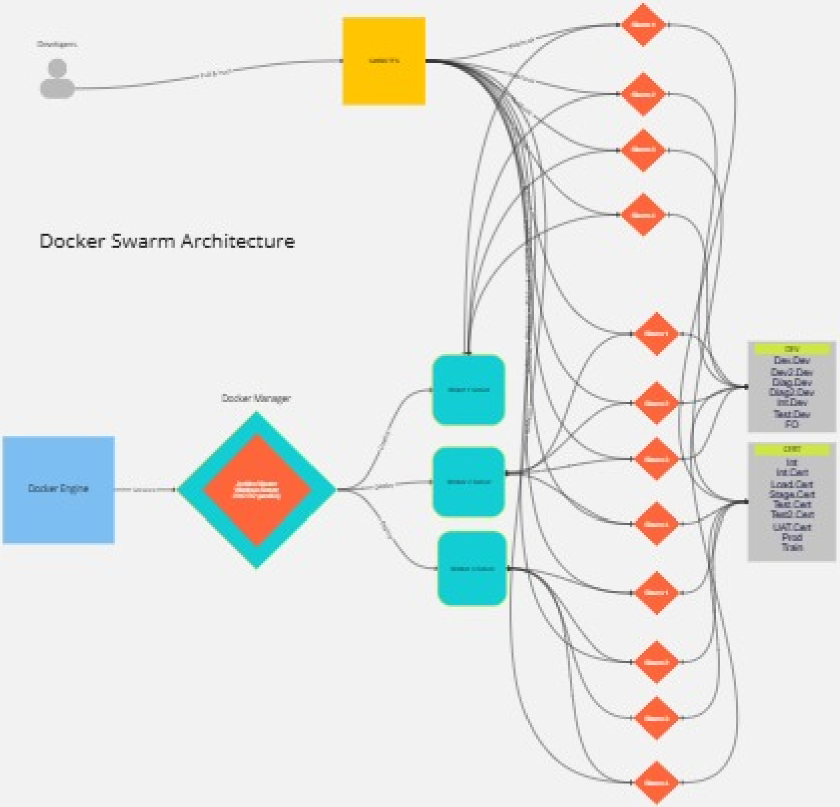
Just need to install docker in the server
That will create all the independencies associated with the system by running docker run.





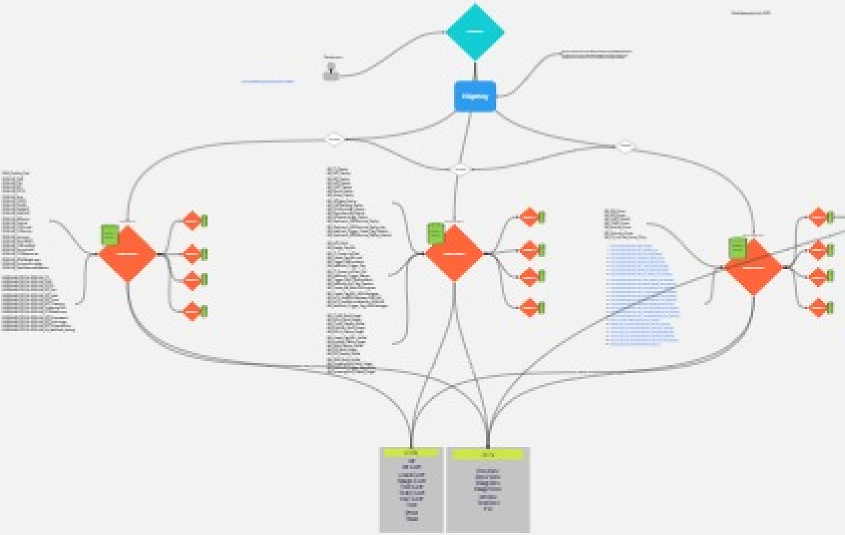
Docker Swarm is a technique to create and maintain a cluster of Docker Engines.
Service deployed in any node can be accessed on other nodes in the same cluster.

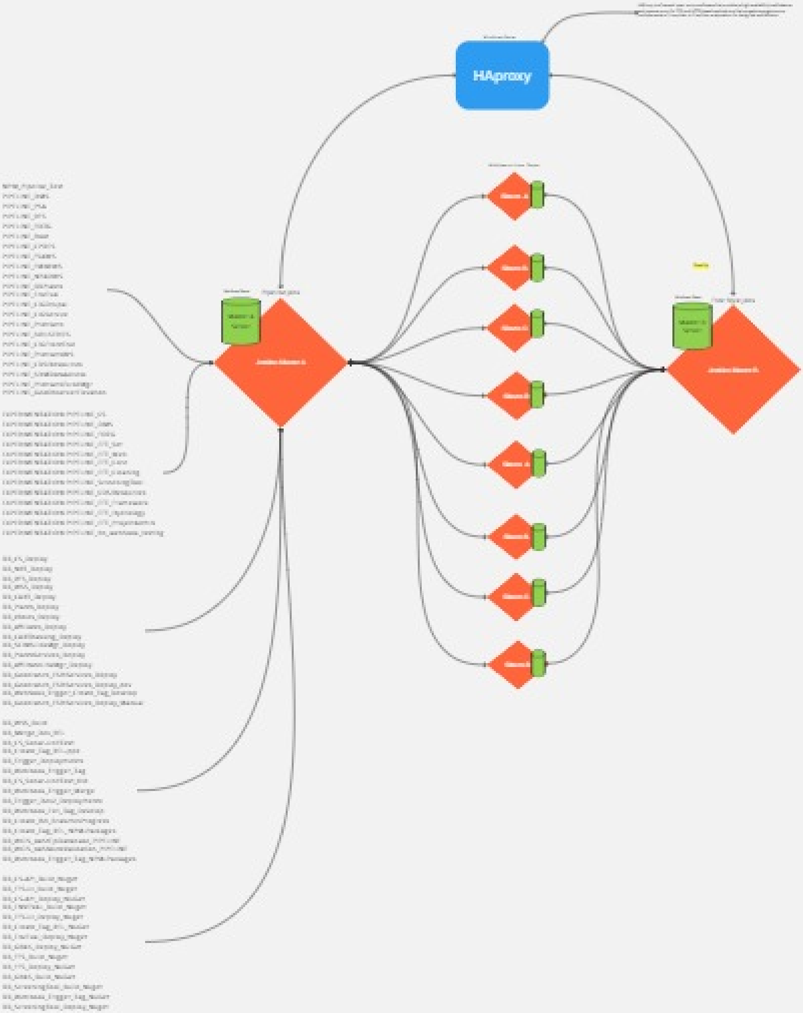
Docker Swarm Architecture



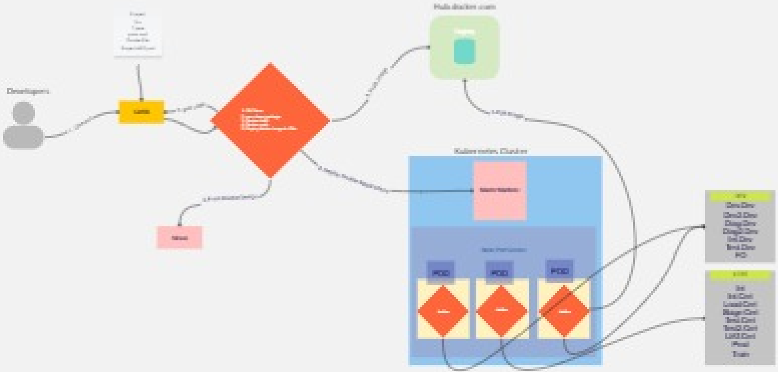
Jenkins Master master architecture

Master slave architecture
Share slaves between master
Share security





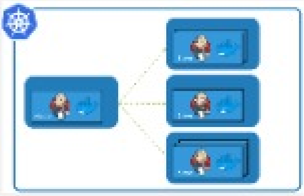
HAproxy


Developers
Hub.docker.com
Kubernetes Cluster
POD
POD
POD

Deploying and Scaling Jenkins on Kubernetes

Jenkins scalability provides many benefits:

- Running many build plans in parallel
- Automatically spinning up and removing agents to save costs
- Distributing the load



To start with, ensure that the Kubernetes Cluster is up and running.
Run this command to view the cluster:
$ kubectl get svc
The output should look similar to this:

Follow the mentioned steps to get this up and running:

- Clone the Github repo and build the docker image.
- Deploy Jenkins helm chart to Kubernetes.
- Access Jenkins.
- Jenkins Slaves Configuration

1. Clone the Github repo and build the docker image:-

First clone the Github repo. This repository has a Dockerfile and a helm chart for setting up a Jenkins master running in Kubernetes. This Jenkins has the required tools to work in and with Kubernetes.

- Jenkins application with pre-loaded plugins (see plugins.txt)
- You can add and remove plugins by editing the plugins.txt file

```
# Build the Jenkins Docker image
$ docker build -t anuphnu/jenkins:v0.0.1 .# Push the image
$ docker push anuphnu/jenkins:v0.0.1
```

2. Deploy Jenkins helm chart to Kubernetes:-

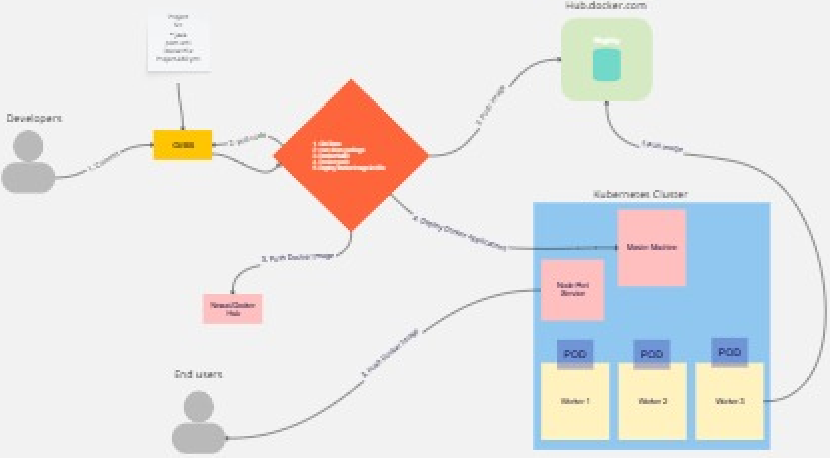
Now we are done with the build image part, it's time to install the Helm.
What is a helm?

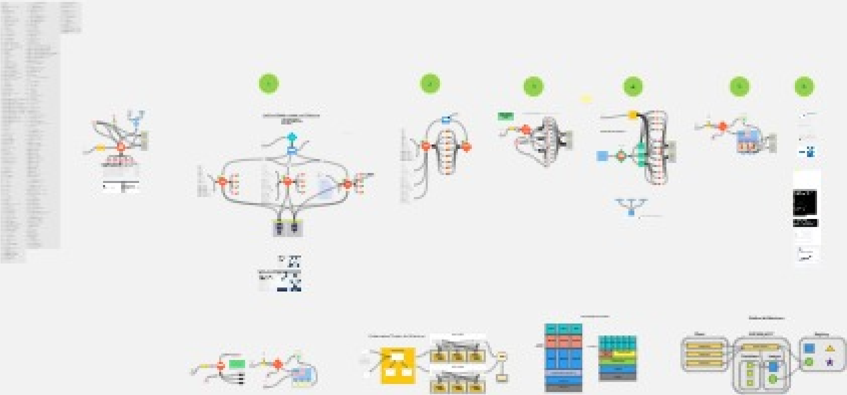
Helm is a package manager which automates the process of installing, configuring, upgrading, and removing complex Kubernetes application. For deployment, you need Kubernetes commands (kubectl) to create and configure resources using Kubernetes manifest. Basically it's manually creating each resource separately which is painful. A Helm chart defines several Kubernetes resources as a set. Helm can make deployments easier and repeatable because all resources for an application are deployed by running one command.
Helm has two elements, a client (helm) and a server (Tiller). The server element runs inside a Kubernetes cluster and manages the installation of charts. With Helm, configuration settings are kept in values.yaml file separate from the manifest formats. The configuration values can be changed according to application need without touching the rest of the manifest.
Install and Enable helm in your cluster:

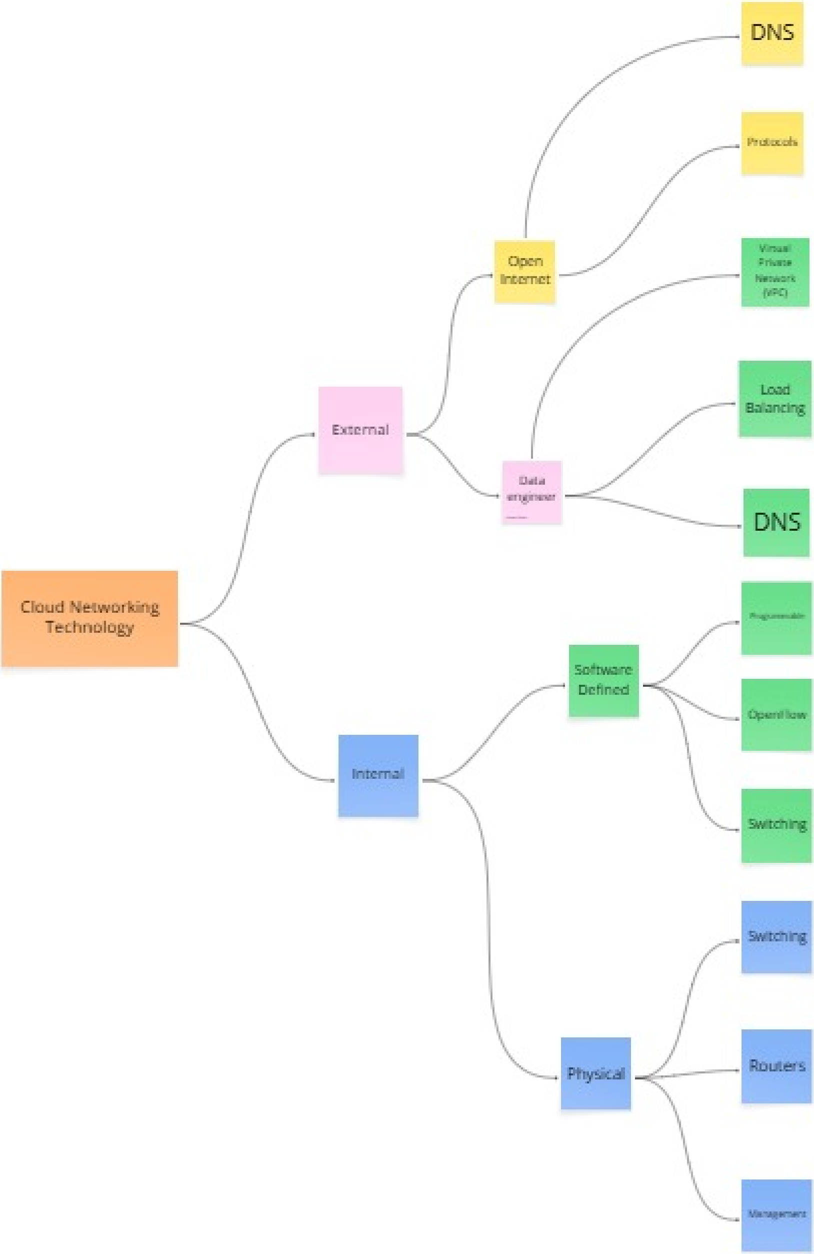
```
# Download helm package and unpack it
$ wget https://get.helm.sh/helm-v3.0.0-rc.2-linux-amd64.tar.gz
$ tar zxfv helm-v3.0.0-rc.2-linux-amd64.tar.gz
$ cp linux-amd64/helm /usr/local/bin/helm# Create The Tiller Service Account and rbac permission
$ kubectl apply -f rbac-config.yaml# Init helm and tiller on your cluster
$ helm init --service-account tiller --upgrade
```

- Deploy the Jenkins helm chart:-

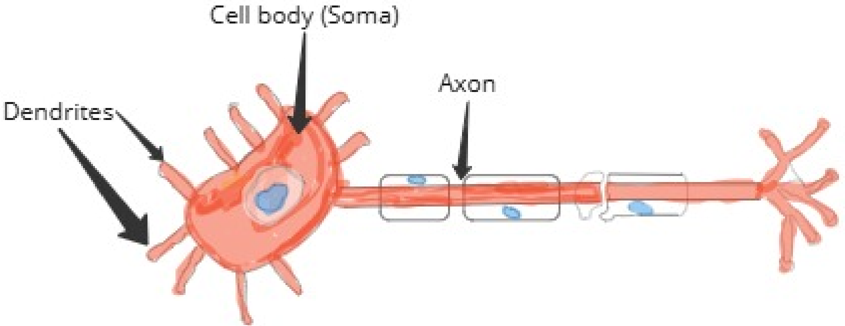
Run the following command to install Jenkins on the Kubernetes cluster via Helm Chart.


Hub.docker.com
Developers
1. Commit
3. Push Docker Image
Nexus/Docker Hub
4. Deploy Docker Application
Kubernetes Cluster
Master Machine
Node Port Service
POD
POD
POD
Worker 1
Worker 2
Worker 3
End users




Cloud Networking Technology
External
Open Internet
DNS
Protocols
Data engineer
Virtual Private Network (VPC)
Load Balancing
DNS
Internal
Software Defined
Programmable
OpenFlow
Switching
Physical
Switching
Routers
Management

Biological Neuron Structure


Cell body (Soma)
Axon
Dendrites