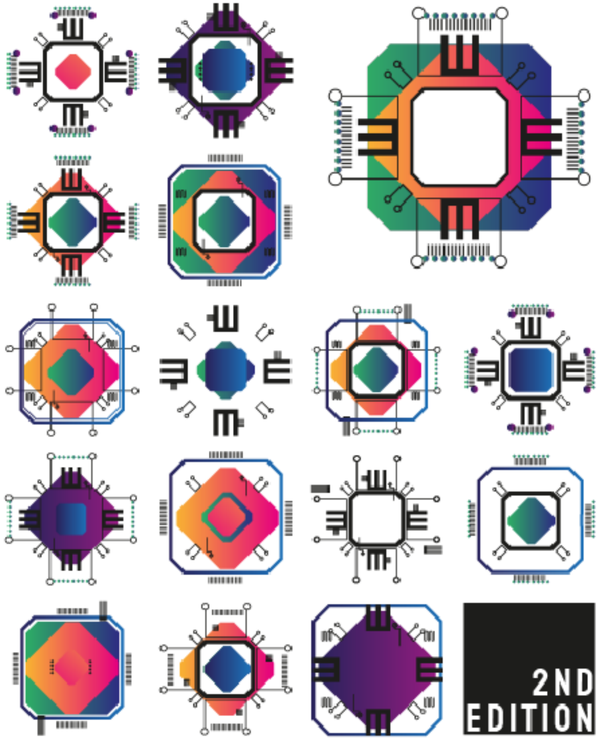


download

INNOVATION

IT Conference & Festival

MAKE DEVELOPERS HAPPY WITH A KUBERNETES CLUSTER

Simone Rota
Senior Software Developer

SORINT.lab

Hello

Simone Rota

Software Developer

Team Leader

@Sorint.lab

Languages I loved through the years:

Little Simon

- Basic
- Pascal (Turbo!)
- Delphi

Scripting Simon

- Perl
- PHP
- Python

9-to-5 Simon

- Java
- SQL

Fun time Simon

▶ Go

What about you?

How many...

...developers?

. . . devops?

...sysadmins?

...none of the above?

So how many

of you have...

...used CI?

...*set up* CI?

...used Docker?

...used Kubernetes?

In this talk I'll try to:

- Convince you CI and containerization understanding is needed
- Present a set of tools to set up a small infrastructure
- Create our infrastructure
- Show it works: code ⇒ deployment

Let's start

Chapter 1: why?

Agile ⇒ CI ⇒ Containers

Agile (or whatever):

- Short, recurring dev cycles
  - Frequent releases
  - Frequent tests
  - Frequent deployments

Automation is required to keep up,

ergo Agile ⇒ CI

What we need for a better CI

- Fast startup times
- Standardization
- Easy release
- Fast release

A container-based infrastructure

is great for this

ergo CI ⇒ Containers

That's why we're gradually moving everything

to containers (plus other reasons, ie scalability)

Who should know about CI & Containers?

Everybody: devs, ops, sys, SREs

Suggestion: create your own infrastructure. Now.

- Keep it simple
- Improve later
- Know the basic steps
- Know how it works

Why should a developer create his own CI/CD infrastructure?

If production is CB, your product is not:

- a war / jar / ear
- a binary
- a script

...it is a Docker image.

⇒ know your sh*t

```
// Datetime info about expense.
loc, err := time.LoadLocation(config.Config.Server.Location)
if err != nil {
    internalServerErrorWithError(err)
    return http.StatusInternalServerError
}
```

😀 Works on my machine

(loc initialized to 'Europe/Rome')

😖 Error 500 on production

(loc is null)

Production has static binary + alpine

...alpine does not ship timezone

```
// LoadLocation returns the Location with the given name.
//
// If the name is "" or "UTC", LoadLocation returns UTC.
// If the name is "Local", LoadLocation returns Local.
//
// Otherwise, the name is taken to be a location name corresponding to a file
// in the IANA Time Zone database, such as "America/New_York".
//
// The time zone database needed by LoadLocation may not be
// present on all systems, especially non-Unix systems.
// LoadLocation looks in the directory or uncompressed zip file
// named by the ZONEINFO environment variable, if any, then looks in
// known installation locations on Unix systems,
// and finally looks in $GOROOT/lib/time/zoneinfo.zip.
```

Could have been easily spotted

The fix (Dockerfile):

```
ENTRYPOINT ["/app/omitted"]
RUN apk add tzdata
```

Best way to get good at packaging your application?

Package somebody else's

⇒ Let's start from our infra

What if we're not using Kubernetes in production (yet)?

Stay ahead!

"I know where this is going"

Be independent

"Dad, I need a vm to test my app, please!"

Be ready from sprint 1!

"First implementation is already online"

Chapter 2: what?

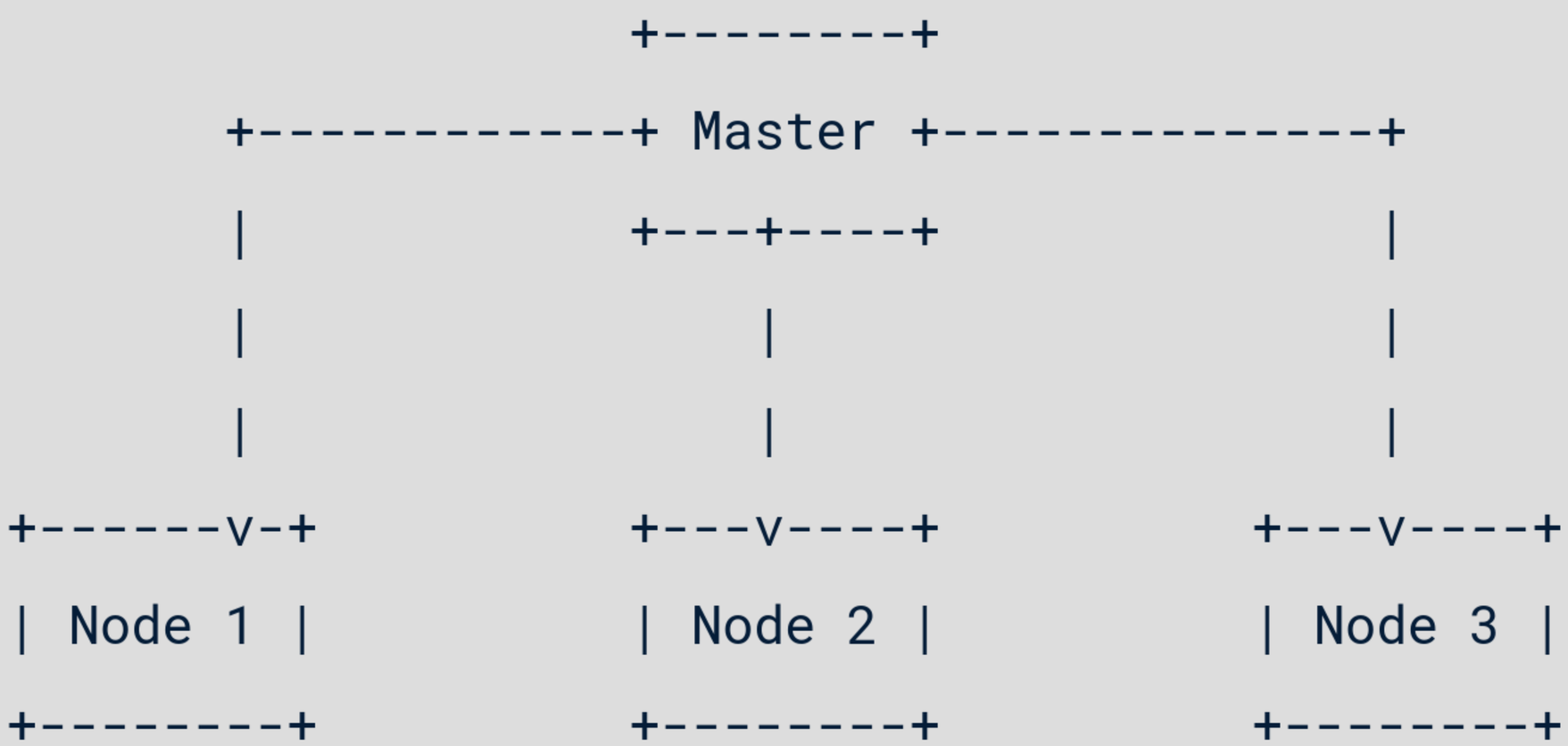
My minimal CI/CD environment:

- Kubernetes Cluster
- Git (Gitea)
- Pipeline (Jenkins)
- QA (Sonar)
- Artifact repo (Nexus)

Prerequisites

Image by Loadmaster (David R. Tribble) - GNU FDL

We assume you have a working Kubernetes cluster. Roll your own (Download 2020?)


Master
Node 1
Node 2
Node 3

Storage

go simple with a NFS share.

Protip #1

register a domain. You're a strong, independent developer.

Protip #2

Abuse the wildcard entry

A k8s.mydomain.com <ip>

CNAME *.mydomain.com -> k8s.mydomain.com

Protip #3

You can use private subnets on public dns

A k8s.mydomain.com 10.210.0.55 (master ip)

Protip #4

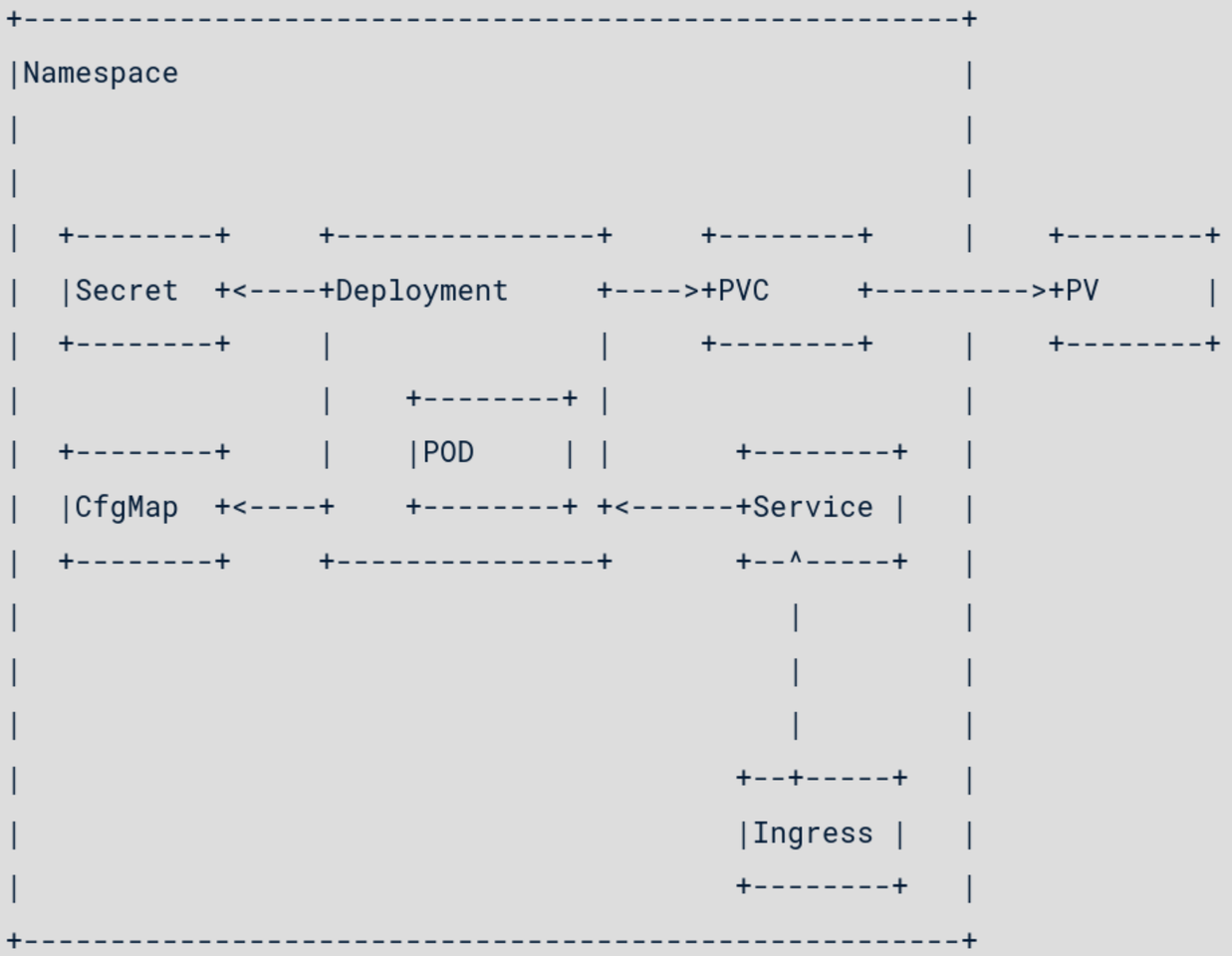
Use SSL from the beginning (Let's Encrypt)

Protip #5

Set up haproxy on the master

Chapter 3: How?

K8S Objects we need


Namespace
Secret
Deployment
PVC
PV
CfgMap
POD
Service
Ingress

PersistentVolume

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pv-postgres
spec:
  storageClassName: local
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  hostPath:
    path: "/k8s/postgres"
```

PersistentVolumeClaim

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: claim-postgres
  namespace: ci
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: local
  resources:
    requests:
      storage: 5Gi
  volumeName: "pv-postgres"
```

Deployment

```
containers:     (extract)
- image: postgres:9.6-alpine
  imagePullPolicy: IfNotPresent
  name: postgres
  ports:
  - containerPort: 5432
    protocol: TCP
  volumeMounts:
  - name: postgres-persistent-storage
    mountPath: /var/lib/postgresql/data
volumes:
  - name: postgres-persistent-storage
    persistentVolumeClaim:
      claimName: claim-postgres
```

Service

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: postgres
  name: postgres
  namespace: ci
spec:
  ports:
  - port: 5432
    protocol: TCP
  selector:
    app: postgres
  sessionAffinity: None
  type: ClusterIP
```

Ingress

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gitea
  namespace: ci
spec:
  rules:
  - host: code.tozzi.fan
    http:
      paths:
      - backend:
          serviceName: gitea-web
          servicePort: 3000
```

- Install kubectl, add to $PATH
- copy admin file from master into ~/kube/config

Protip #6

install kubectl completion

Command examples

```
# Create namespace
$ kubectl create namespace ci
```

```
# Deploy a tool
$ kubectl create -f pv.yml
$ kubectl create -f pvc.yml
$ kubectl create -f deployment.yml
$ kubectl create -f svc.yml
$ kubectl create -f ingress.yml
```

```
# Check pod status
```

```
$ kubectl -n ci get po
$ kubectl -n ci logs postgres-7996587c-gf9l6
```

```
# Check other objects
$ kubectl -n ci get svc
$ kubectl -n ci get deployments
$ kubectl -n ci get ingress
```

Deploy all the tools

- postgresql
- gitea
- nexus
- sonar
- jenkins

Everything up?

Set up the tools

0. PostgreSQL

```
# Let's go on the pod
$ kubectl -n ci exec -it postgres-7996587c-gf9l6 bash
```

```
# Create database
# su - postgres
$ createuser -P gitea
$ createdb -O gitea gitea
```

1. Gitea

- Follow initial web setup
- Register
- Login
- Create a token
- Create organization

2. Nexus

- Login ⇒ Administration
- Create a dedicated user
- Security ⇒ Realms ⇒ enable "Docker Token Bearer"
- Create maven, binary, docker repos
- Docker repo: check http port, V1 API

3. Sonar

No cfg needed ATM

4. Jenkins

Here comes the fun part

Plugins:

- Gitea Plugin
- Kubernetes Plugin
- Pipeline
- Blue Ocean (eye candy)

Credentials:

- gitea access Token
- nexus username & password
- k8s certificate
- sonar username & password

Configuration:

- Gitea server
- Kubernetes
- Sonar instance

Create a gitea organization item in Jenkins.

Jenkins generates the the gitea webhooks to handle push, pull requests, etc. for all projects.

The hard part is over, now it's all handled in the projects repos

5. Your projects

- Create Dockerfile
- Create Jenkinsfile
- Create k8s files

Dockerfile

```
FROM alpine:3.10.2
COPY api-go /app/
ENTRYPOINT ["/app/api-go"]
```

k8s files

```
# Deployment
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: api-go
  name: api-go
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: api-go
  template:
    metadata:
      labels:
        app: api-go
    spec:
      containers:
      - image: registry.tozzi.fan/api-go
        imagePullPolicy: Always
        name: api-go
        ports:
        - containerPort: 8080
          protocol: TCP
      restartPolicy: Always
```

```yaml
# Service
apiVersion: v1
kind: Service
metadata:
  namespace: default
  labels:
    app: api-go
  name: api-go
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: api-go
  type: ClusterIP
```

```
# Ingress
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: api-go
  namespace: default
spec:
  rules:
  - host: api-go.tozzi.fan
    http:
      paths:
      - backend:
          serviceName: api-go
          servicePort: 8080
```

Jenkinsfile

Example pipeline:

- Checkout
- Build
- Test
- Publish Artifact
- Build Docker
- Push Docker
- Deploy to dev env

Jenkinsfile

- Define pod template(s)
- Specify container
- Write your stages

```
podTemplate (
    label: label,
    containers: [
        containerTemplate(name: 'go', image: 'golang:1.12.9',
          ttyEnabled: true, command: 'cat'),
        containerTemplate(name: 'buildah', image: 'sorintdev/buildah:f30.1',
          ttyEnabled: true, command: 'cat')
    ],
    volumes: [
        hostPathVolume(hostPath: '/usr/bin/kubectl', mountPath: '/usr/bin/kubectl'),
        secretVolume(mountPath: '/etc/kubernetes', secretName: 'cluster-admin')],
)
```

```
stage ('Checkout') {
    scmVars = git url: "https://code.tozzi.fan/dreamteam/api-go.git",
      branch: env.BRANCH_NAME
    commit = scmVars.GIT_COMMIT
}
```

```
container('go') {

...

container('buildah') {

...
```

```
stage('Run tests') {
    sh "go test ."
}
```

```
stage('Build with go') {
    parallel(linux: {
        sh "CGO_ENABLED=0 go build -o api-go -a -x"
    },
    windows : {
        sh "GOOS=windows GOARCH=amd64 CGO_ENABLED=0 go build -o api-go.exe -a -x"
    })
}
```

```
stage('Publish artifact(s)') {
    withCredentials([usernamePassword(credentialsId: 'nexus',
          usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD')]) {
        sh "curl -v -k -u ${USERNAME}:${PASSWORD} --upload-file api-go
          https://nexus.tozzi.fan/repository/binaries/api-go/api-go-linux-${version}"
        sh "curl -v -k -u ${USERNAME}:${PASSWORD} --upload-file api-go.exe
          https://nexus.tozzi.fan/repository/binaries/api-go/api-go-linux-${version}.exe"
    }
}
```

```
stage('Build docker image') {
    sh "buildah --format docker --storage-driver vfs
      build-using-dockerfile -t registry.tozzi.fan/api-go:${version} ."
}
```

```
stage('Publish docker image') {
    withCredentials([usernamePassword(credentialsId: 'nexus',
        usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD')]) {
        sh "buildah --format docker --storage-driver vfs
          --creds=${USERNAME}:${PASSWORD} push registry.tozzi.fan/api-go:${version}"
    }
}
```

```
stage ('Kubernetes Deploy') {
    if (branch == 'master') {
        sh "kubectl replace --force -f k8s/service.yml
          --kubeconfig=/etc/kubernetes/kubernetes.conf"
        sh "kubectl replace --force -f k8s/deployment.yml
          --kubeconfig=/etc/kubernetes/kubernetes.conf"
        sh "kubectl replace --force -f k8s/ingress.yml
          --kubeconfig=/etc/kubernetes/kubernetes.conf"
    }
}
```

Chapter 4: results

Showtime

Chapter 5. next steps

- Enhance security: k8s user/role bindings
- Handle more namespaces (dev, test, etc.)
- Adjust pipeline to your git workflow
- Handle more namespaces (dev, test, etc.)
- ...and much more

Repo with samples at

https://github.com/simonerota/kubernetes-ci-for-devs

Shameless plug...

https://agola.io/

CI/CD redefined

Jenkins ⇒ Agola

Talk tomorrow at 11:00

by Simone Gotti

Thanks!