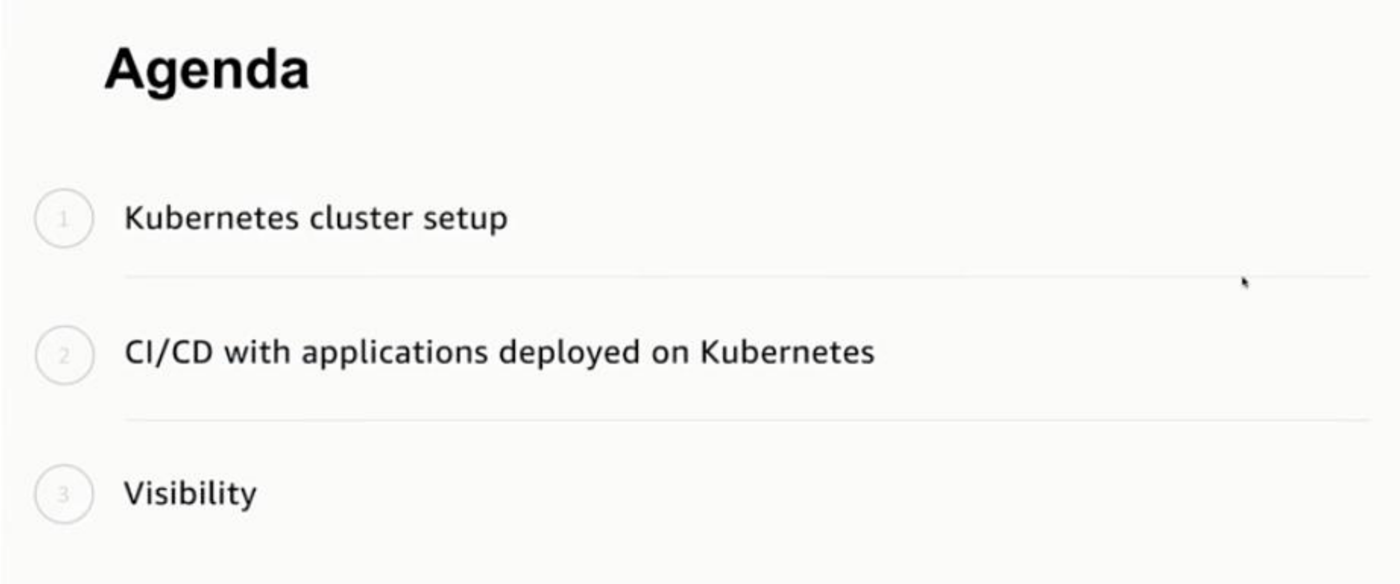
AWS SUMMIT TEL AVIV

Mastering Kubernetes on AWS

Arun Gupta, @arungupta
Principal Open Source Technologist, AWS

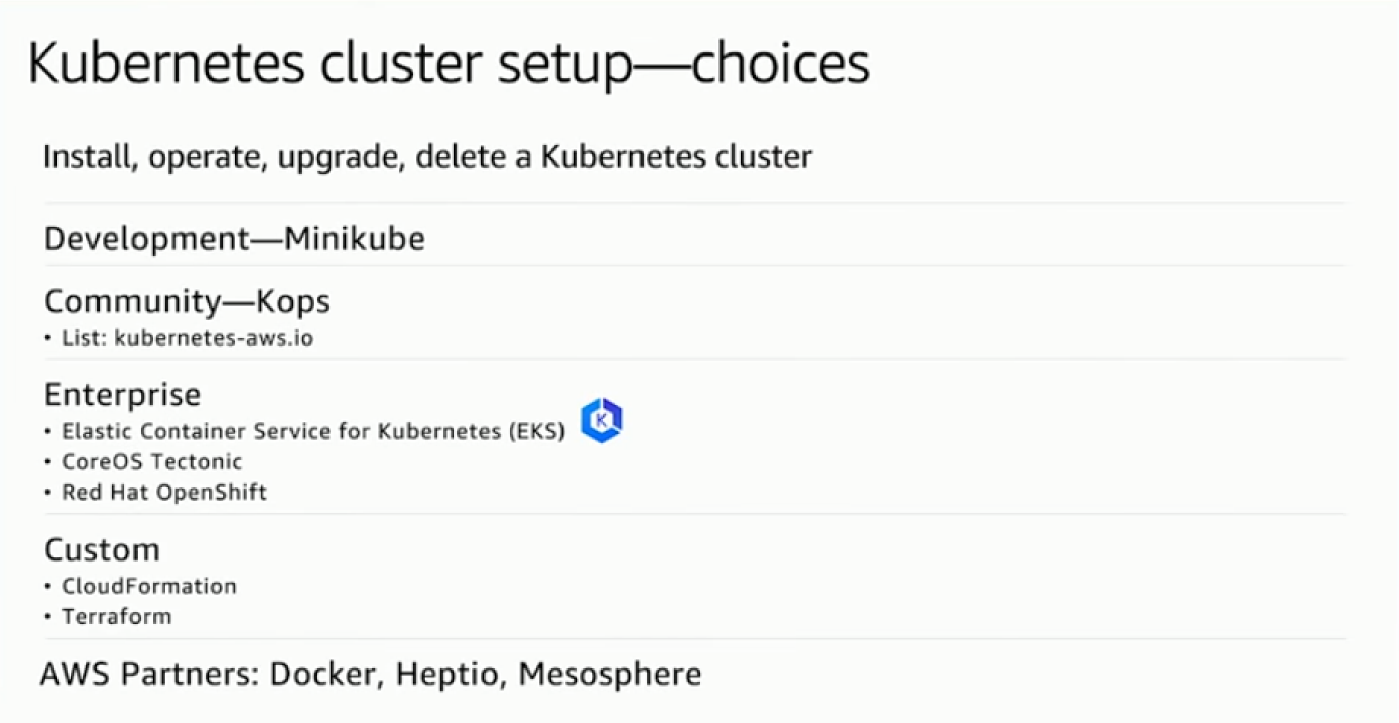
Amit Bezalel @amitbez
Chief System Architect, Microfocus
(github.com/amitbet)



Agenda

1. Kubernetes cluster setup
2. CI/CD with applications deployed on Kubernetes
3. Visibility

1 Kubernetes cluster set-up choices



Kubernetes cluster setup—choices

Install, operate, upgrade, delete a Kubernetes cluster

Development—Minikube

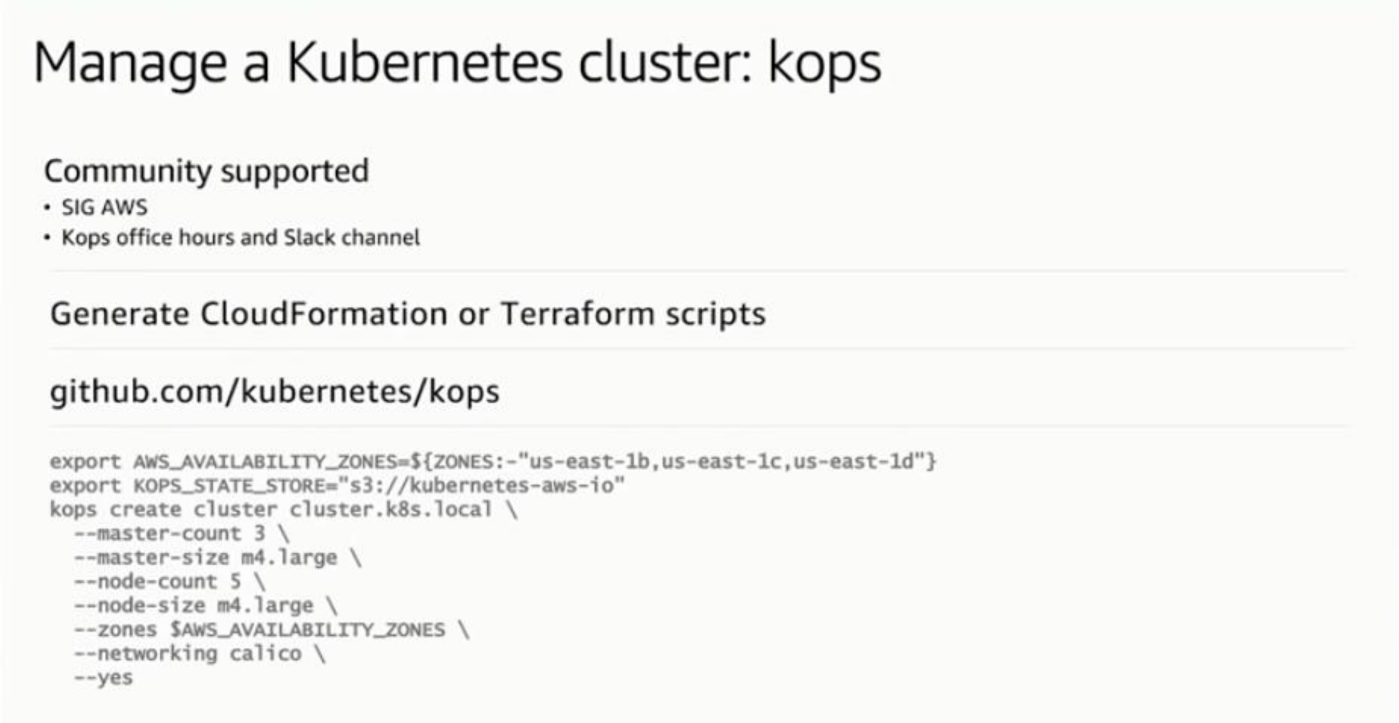
Community—Kops
- List: kubernetes-aws.io

Enterprise
- Elastic Container Service for Kubernetes (EKS)
- CoreOS Tectonic
- Red Hat OpenShift

Custom
- CloudFormation
- Terraform

AWS Partners: Docker, Heptio, Mesosphere

With EKS, AWS gives you a control plane and you bring your worker nodes and attach to the Master node you will get



Manage a Kubernetes cluster: kops

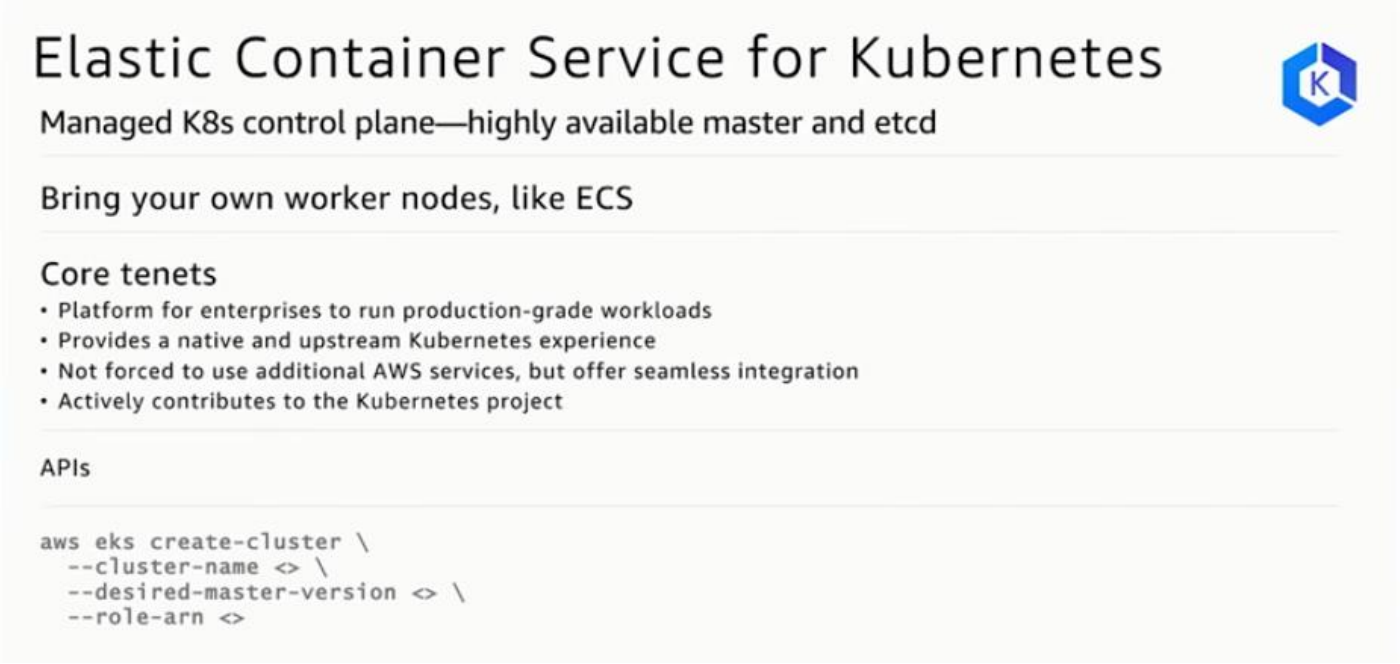
Community supported
- SIG AWS
- Kops office hours and Slack channel

Generate CloudFormation or Terraform scripts

github.com/kubernetes/kops

```
export AWS_AVAILABILITY_ZONES=${ZONES:-"us-east-1b,us-east-1c,us-east-1d"}
export KOPS_STATE_STORE="s3://kubernetes-aws-io"
kops create cluster cluster.k8s.local \
  --master-count 3 \
  --master-size m4.large \
  --node-count 5 \
  --node-size m4.large \
  --zones $AWS_AVAILABILITY_ZONES \
  --networking calico \
  --yes
```

Above is the command that you can use to create a K8s cluster using KOPS in 3 AZs, you specify the parameters above and the state of the cluster itself will be stored in a S3 bucket



Elastic Container Service for Kubernetes

Managed K8s control plane—highly available master and etcd

Bring your own worker nodes, like ECS

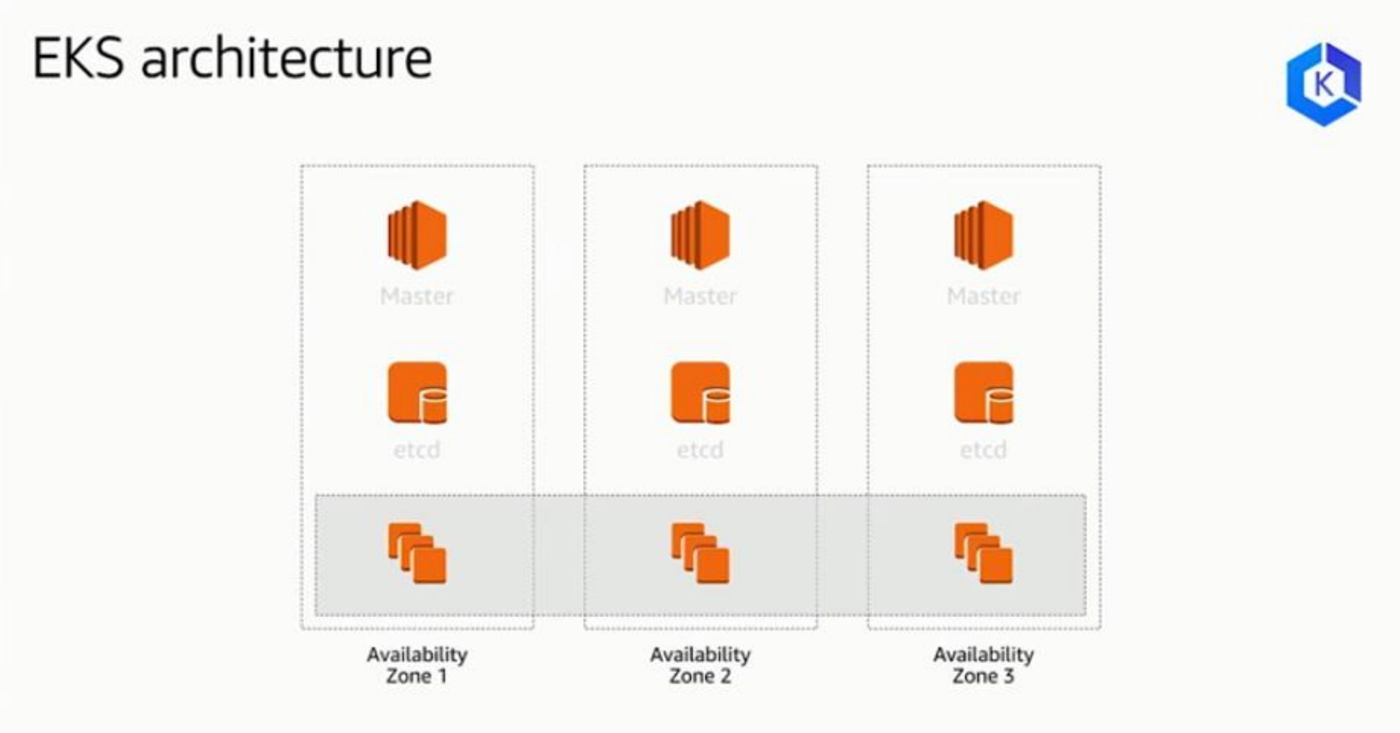
Core tenets

- Platform for enterprises to run production-grade workloads
- Provides a native and upstream Kubernetes experience
- Not forced to use additional AWS services, but offer seamless integration
- Actively contributes to the Kubernetes project

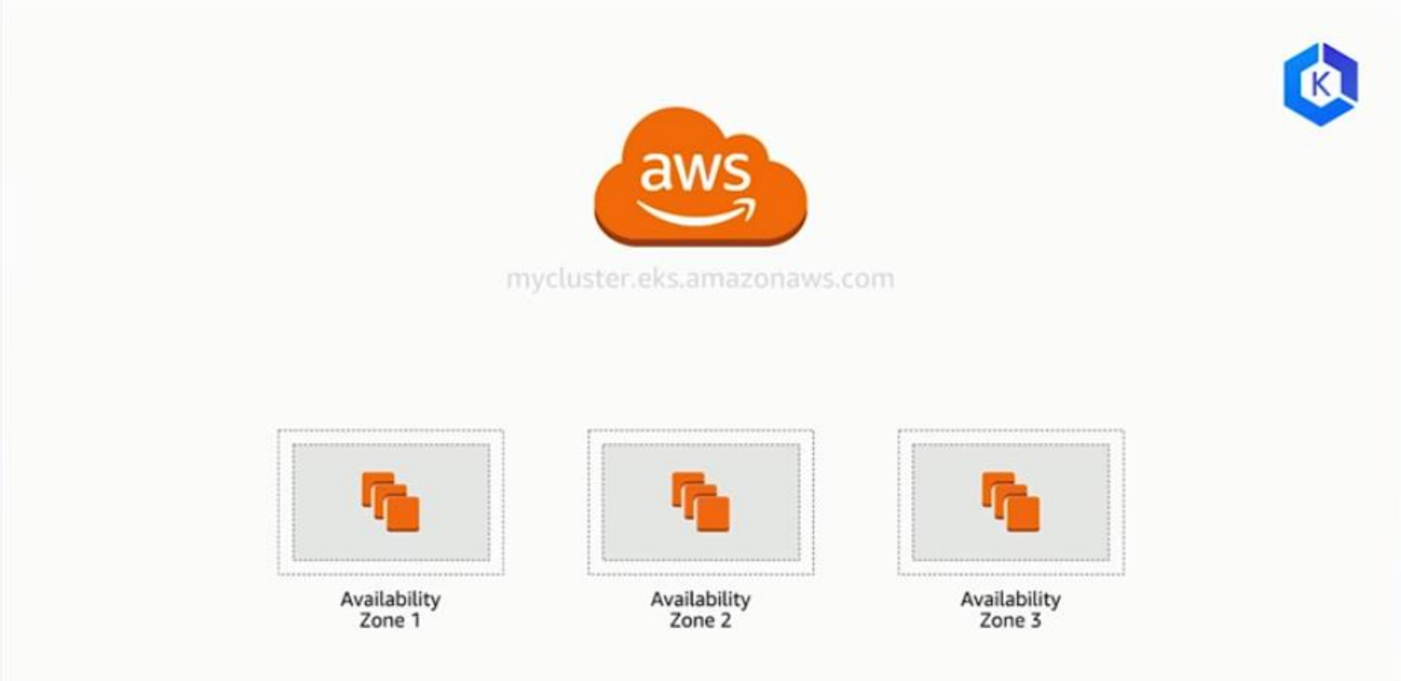
APIs

```
aws eks create-cluster \
  --cluster-name <> \
  --desired-master-version <> \
  --role-arn <>
```

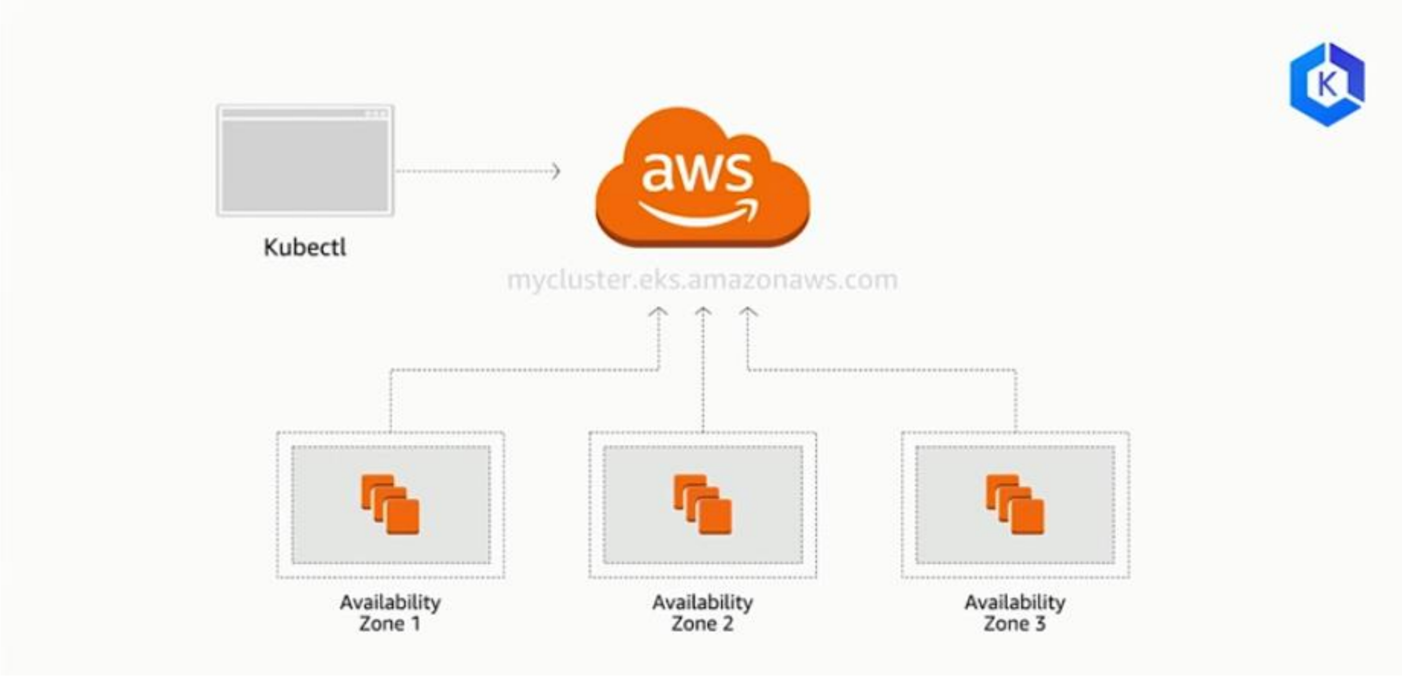
You need to give the EKS K8s cluster an IAM role that it will use when creating those EC2 instances



EKS architecture



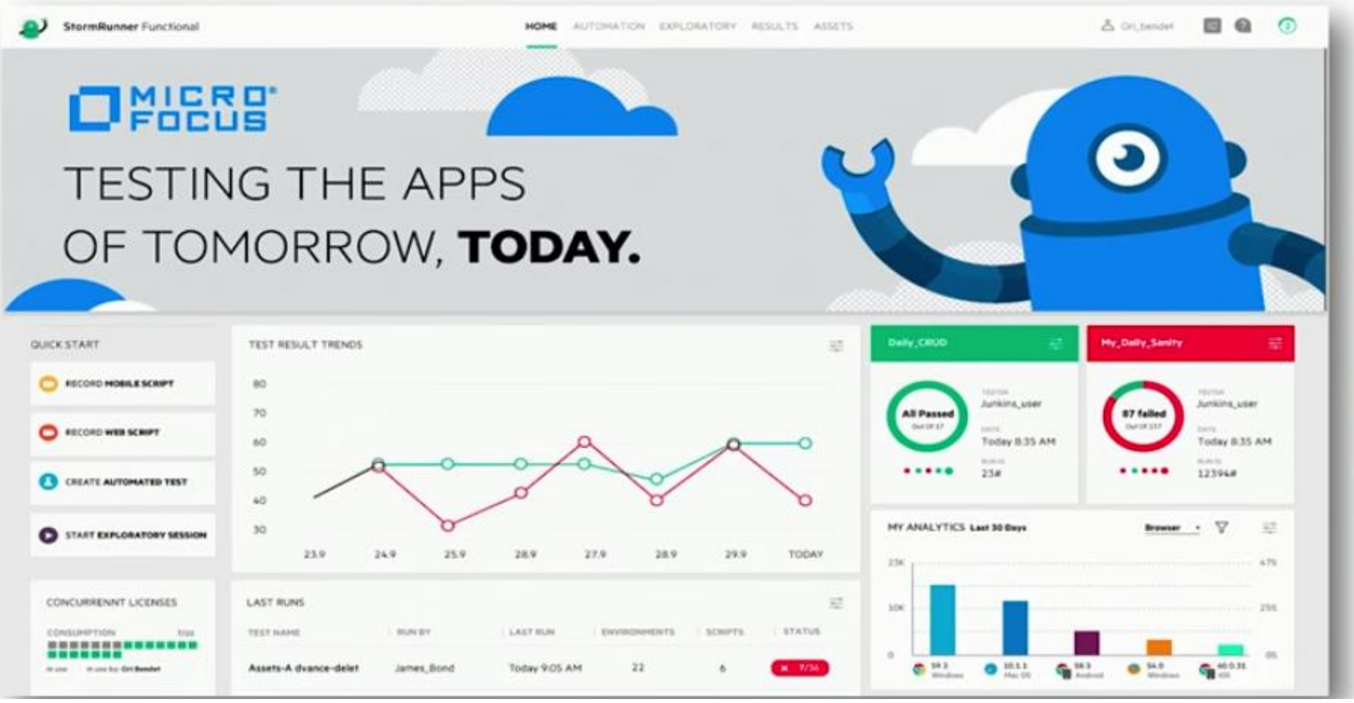
EKS abstracts away the control plane from you and instead gives you a control plane that includes an API server, Controller, Scheduler (all the key components of the K8s master), you juts need to bring your worker nodes



You then attach your worker nodes to the Master/control plane, then you can start using ***Kubectl*** API to interact and issue commands to the Master node for creating, managing and controlling your apps

Cluster setup at Microfocus

StormRunner Functional



TESTING THE APPS
OF TOMORROW, **TODAY.**

TESTER	DATE	RUN ID
Junkins_user	Today 8:35 AM	23#
Junkins_user	Today 8:35 AM	12394#

TEST NAME	RUN BY	LAST RUN	ENVIRONMENTS	SCRIPTS	STATUS
Assets-A dvance-delet	James_Bond	Today 9:05 AM	22	6	✕ 7/36

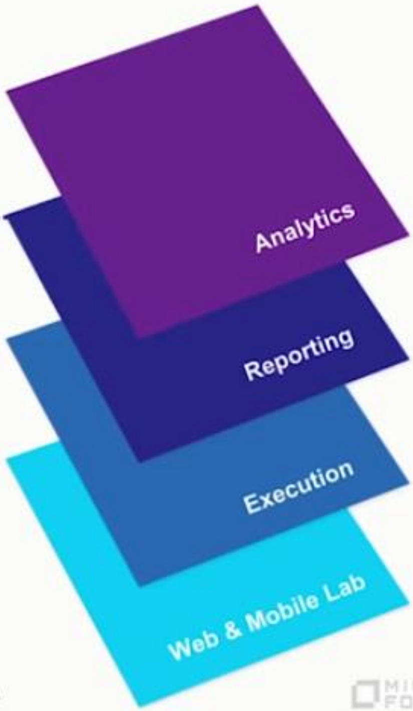
SRF High Level Solution

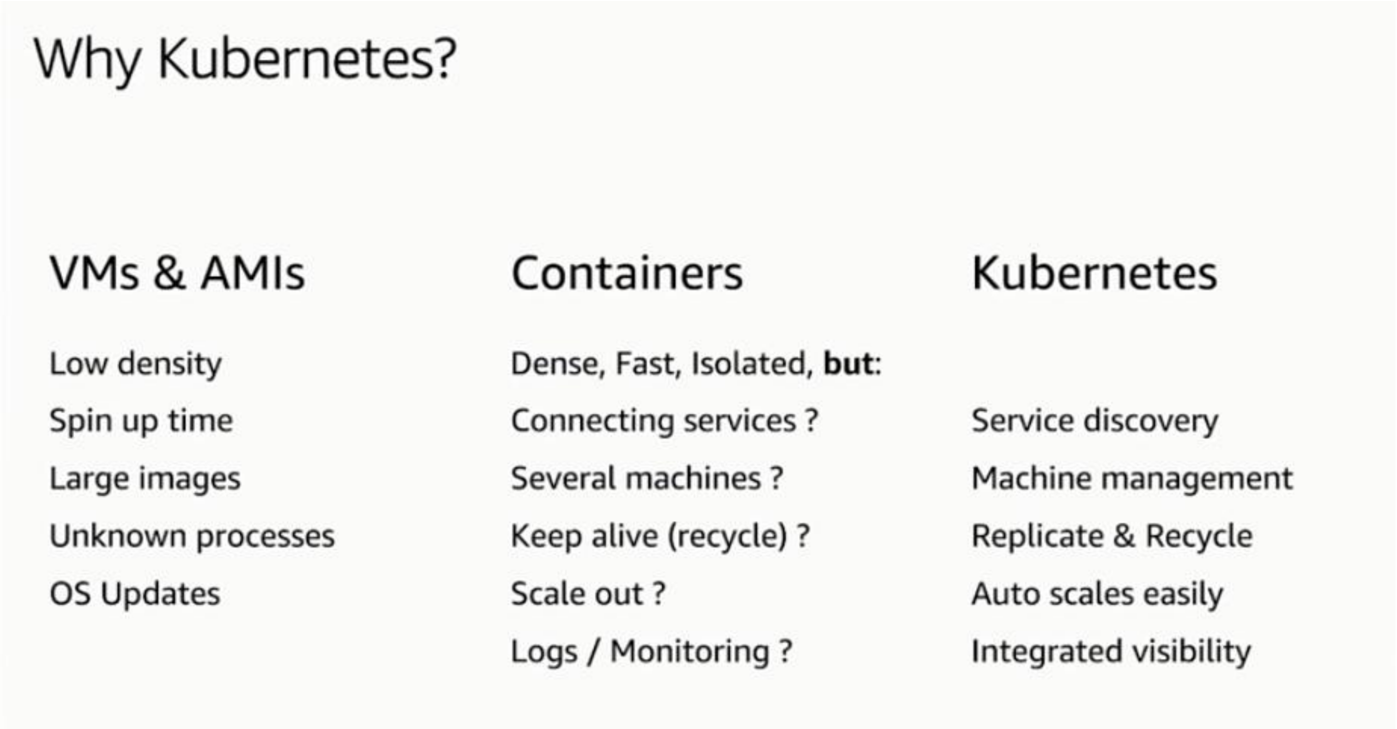
Analytics & Predictive
INTERNAL | PRODUCTION | INDUSTRY

Reporting (RCA)
STATUS | DEFECTS | TRENDS

Execution (Test & Scripts)
REMOTE | UPLOAD | RECORD

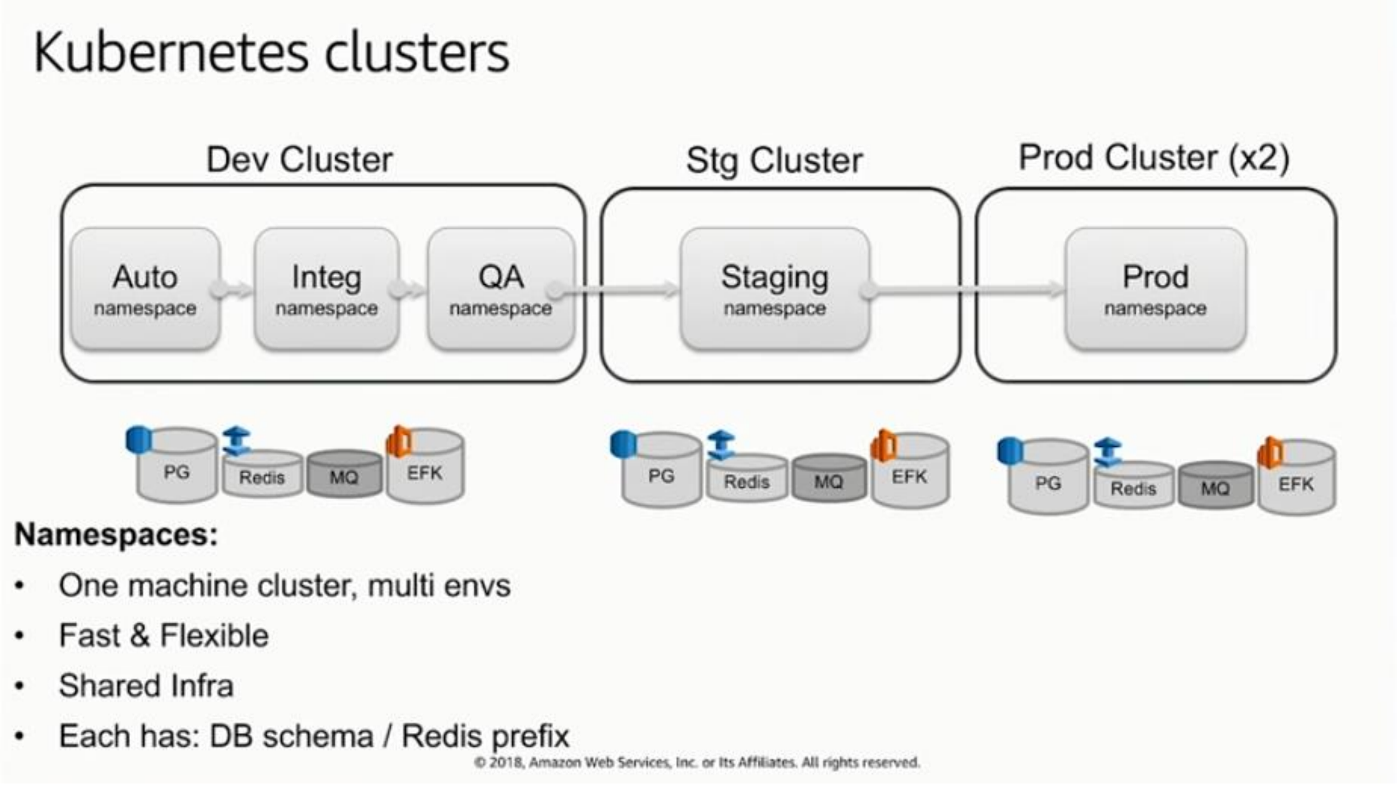
LAB Service (Cloud/On-prem & Hybrid)
WEB | MOBILE | THINGS*





Why Kubernetes?

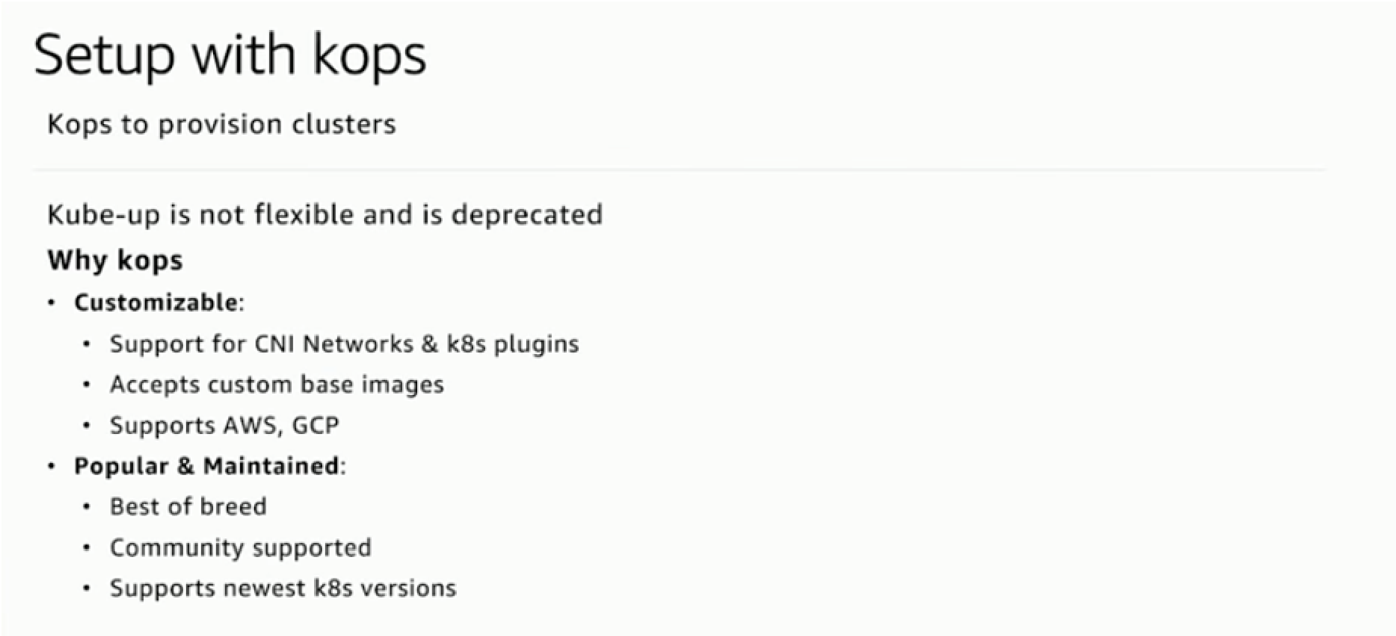
VMs & AMIs	Containers	Kubernetes
Low density	Dense, Fast, Isolated, **but**:	
Spin up time	Connecting services ?	Service discovery
Large images	Several machines ?	Machine management
Unknown processes	Keep alive (recycle) ?	Replicate & Recycle
OS Updates	Scale out ?	Auto scales easily
	Logs / Monitoring ?	Integrated visibility



Kubernetes clusters

Namespaces:

- One machine cluster, multi envs
- Fast & Flexible
- Shared Infra
- Each has: DB schema / Redis prefix



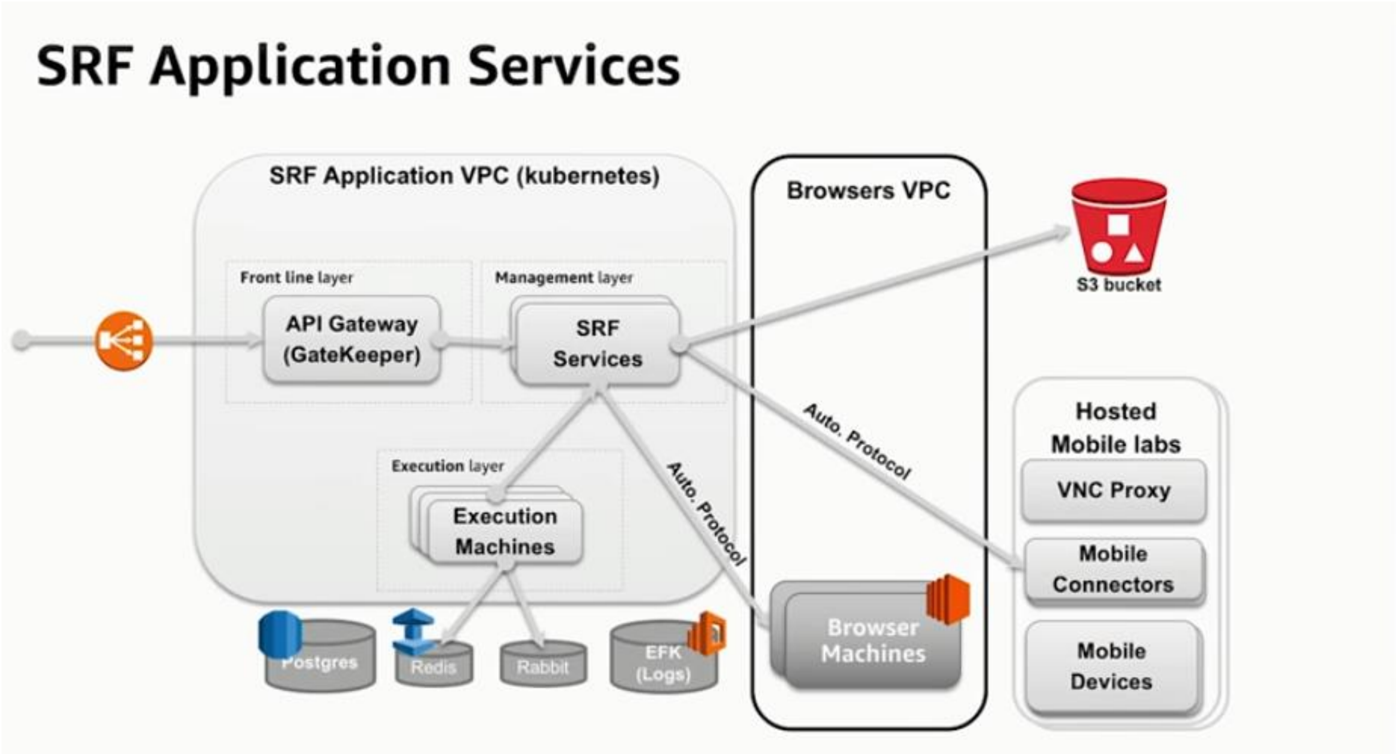
Setup with kops

Kops to provision clusters

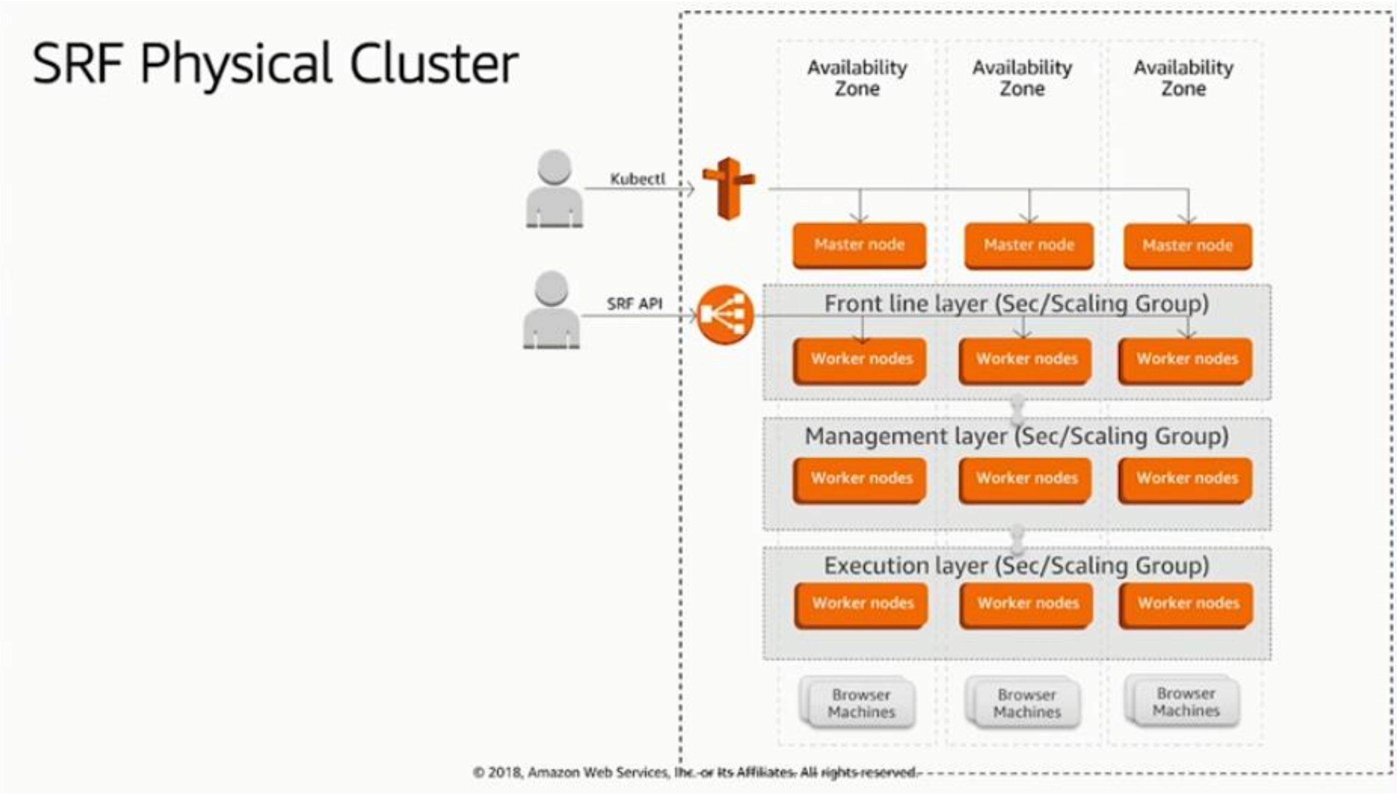
Kube-up is not flexible and is deprecated

Why kops

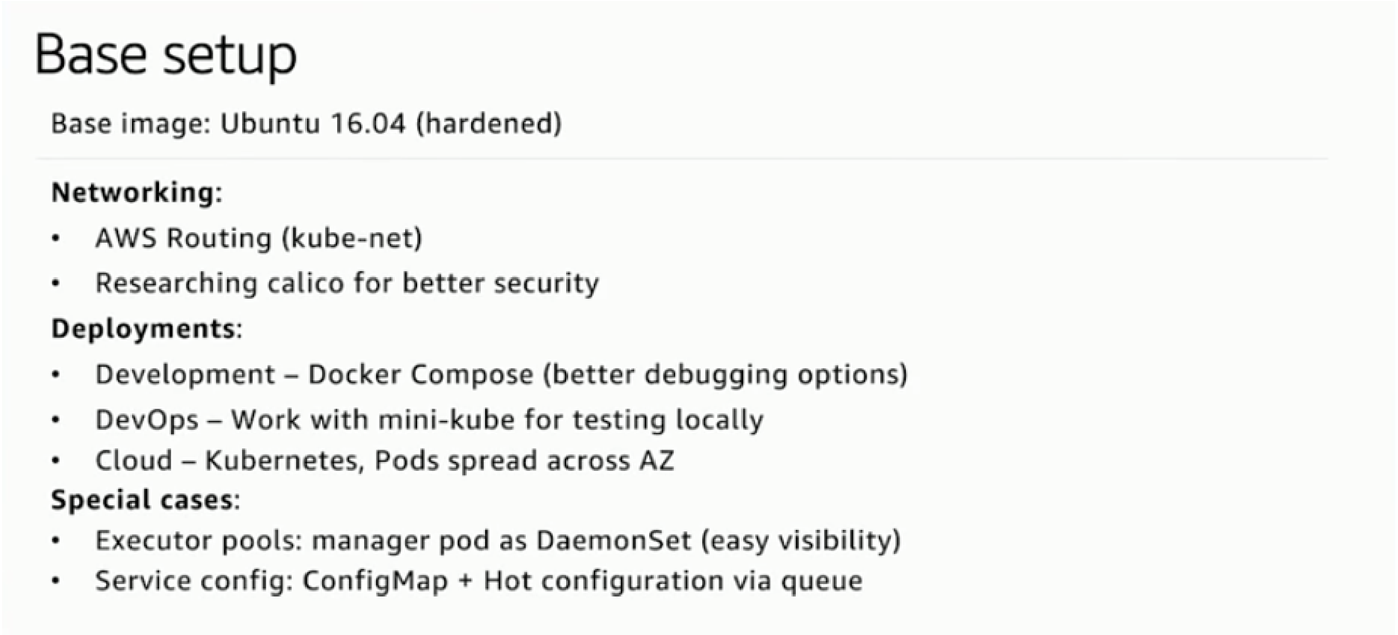
- **Customizable**:
  - Support for CNI Networks & k8s plugins
  - Accepts custom base images
  - Supports AWS, GCP
- **Popular & Maintained**:
  - Best of breed
  - Community supported
  - Supports newest k8s versions



SRF Application Services



SRF Physical Cluster



Base setup

Base image: Ubuntu 16.04 (hardened)

Networking:

- AWS Routing (kube-net)
- Researching calico for better security

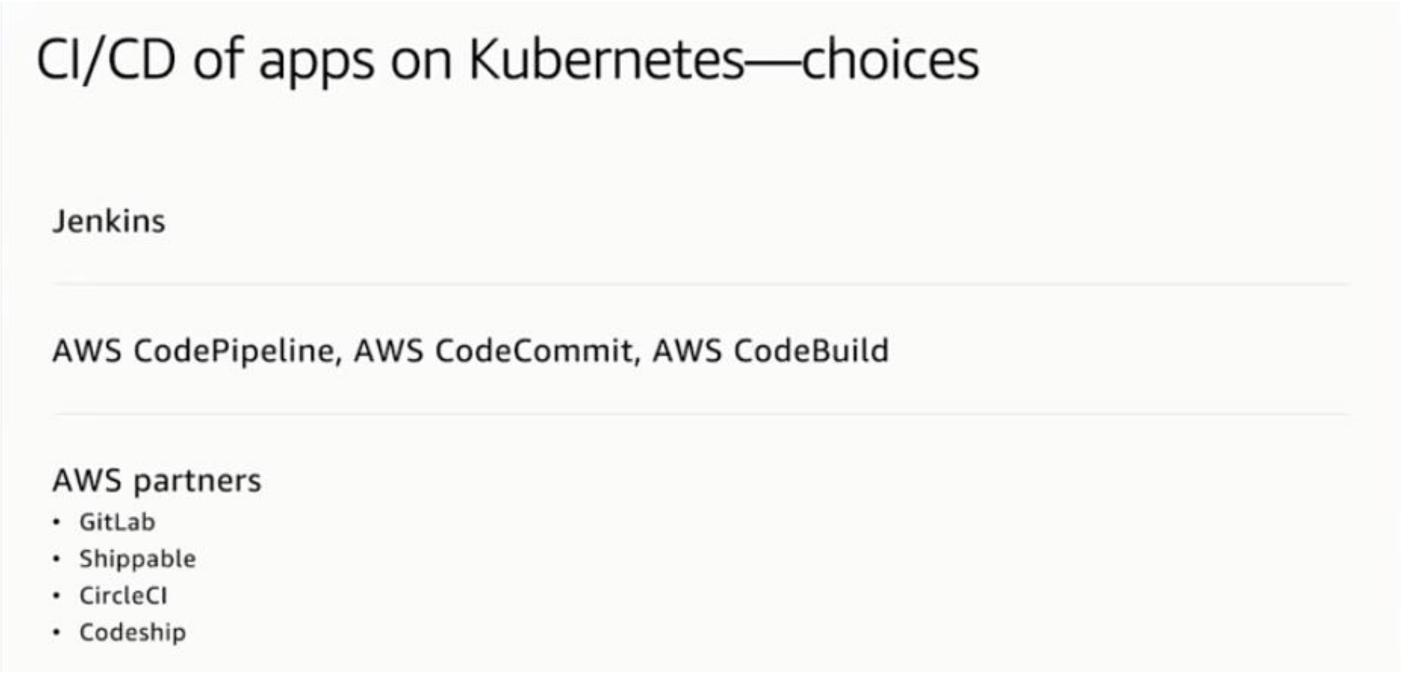
Deployments:

- Development – Docker Compose (better debugging options)
- DevOps – Work with mini-kube for testing locally
- Cloud – Kubernetes, Pods spread across AZ

Special cases:

- Executor pools: manager pod as DaemonSet (easy visibility)
- Service config: ConfigMap + Hot configuration via queue

2 CI/CD of apps on Kubernetes



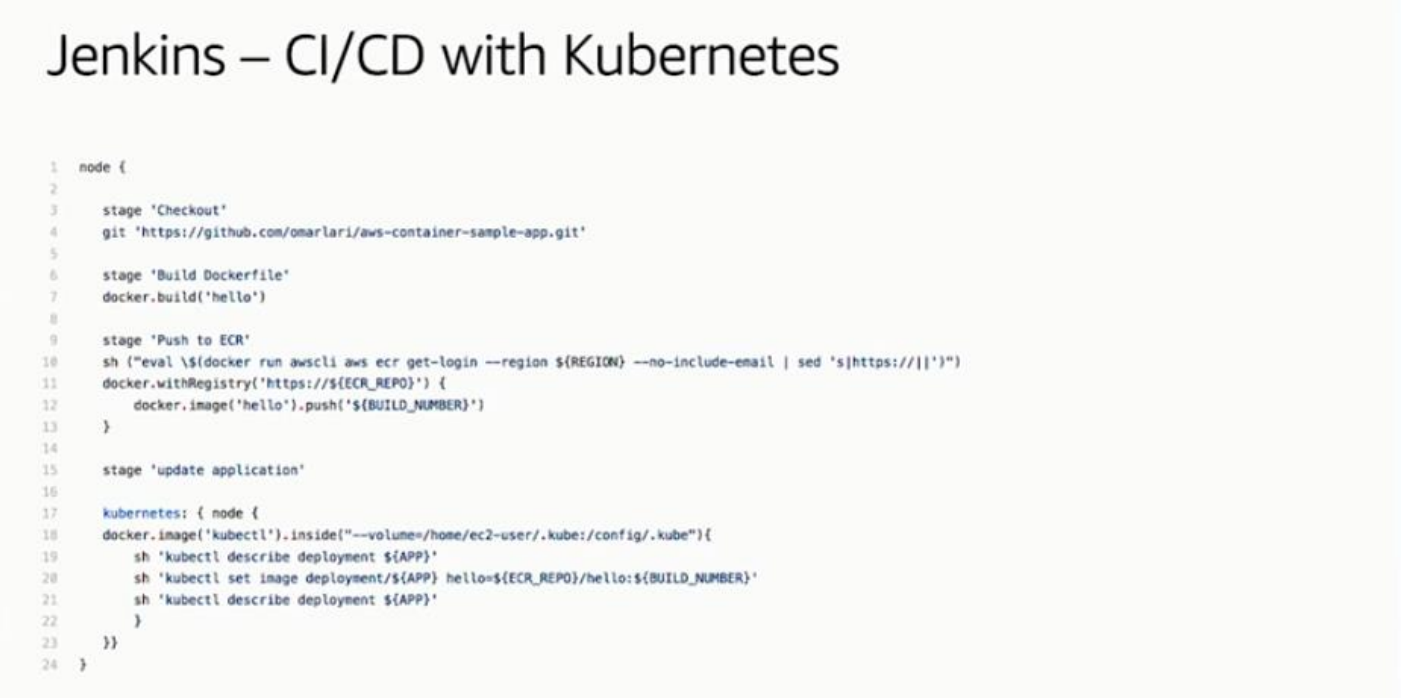
CI/CD of apps on Kubernetes—choices

Jenkins

AWS CodePipeline, AWS CodeCommit, AWS CodeBuild

AWS partners

- GitLab
- Shippable
- CircleCI
- Codeship

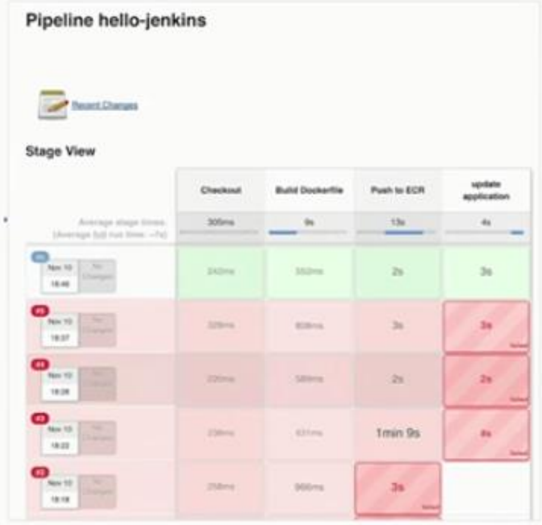


Jenkins – CI/CD with Kubernetes

```
node {

   stage 'Checkout'
   git 'https://github.com/omarlari/aws-container-sample-app.git'

   stage 'Build Dockerfile'
   docker.build('hello')

   stage 'Push to ECR'
   sh ("eval \$(docker run awscli aws ecr get-login --region ${REGION} --no-include-email | sed 's|https://||')")
   docker.withRegistry('https://${ECR_REPO}') {
       docker.image('hello').push('${BUILD_NUMBER}')
   }

   stage 'update application'

   kubernetes: { node {
   docker.image('kubectl').inside("--volume=/home/ec2-user/.kube:/config/.kube"){
       sh 'kubectl describe deployment ${APP}'
       sh 'kubectl set image deployment/${APP} hello=${ECR_REPO}/hello:${BUILD_NUMBER}'
       sh 'kubectl describe deployment ${APP}'
       }
   }}
}
```

Jenkins – CI/CD with Kubernetes

```
node {

  stage 'Checkout'
  git 'https://github.com/omarlari/aws-container-sample-app.git'

  stage 'Build Dockerfile'
  docker.build('hello')

  stage 'Push to ECR'
  sh ("eval \$(docker run awscli aws ecr get-login --region ${REGION} --no-include-email | sed '
  docker.withRegistry('https://${ECR_REPO}') {
      docker.image('hello').push('${BUILD_NUMBER}')
  }

  stage 'update application'

  kubernetes: { node {
  docker.image('kubectl').inside("--volume=/home/ec2-user/.kube:/config/.kube"){
      sh 'kubectl describe deployment ${APP}'
      sh 'kubectl set image deployment/${APP} hello=${ECR_REPO}/hello:${BUILD_NUMBER}'
      sh 'kubectl describe deployment ${APP}'
      }
  }}
}
```



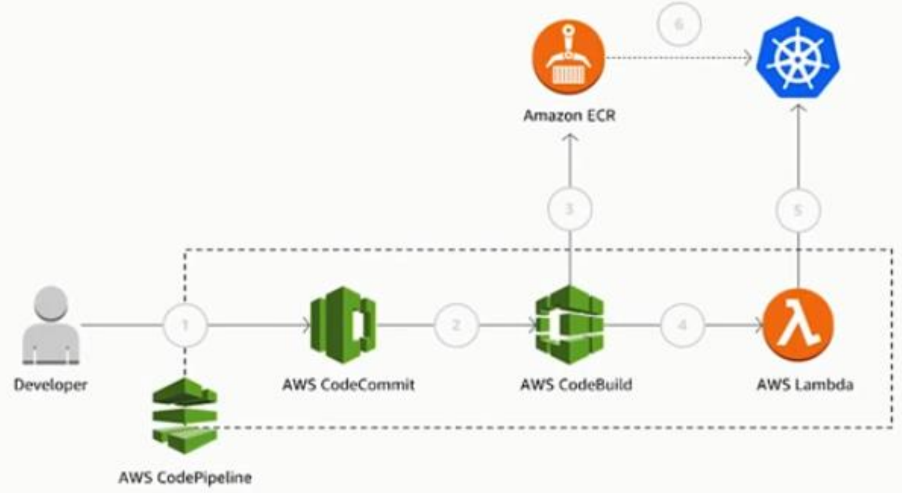
Pipeline hello-jenkins

Recent Changes

Stage View

	Checkout	Build Dockerfile	Push to ECR	update application
Average stage times: (Average full run time: [illegible])	305ms	9s	13s	4s
Nov 10 18:46 No Changes	240ms	350ms	2s	3s
Nov 10 18:37 No Changes	329ms	808ms	3s	3s
Nov 10 18:28 No Changes	220ms	589ms	2s	2s
Nov 10 18:22 No Changes	236ms	631ms	1min 9s	4s
Nov 10 18:18 No Changes	258ms	966ms	3s	

AWS CodePipeline – CI/CD with Kubernetes



1 Developers continuously integrate changes into a main branch hosted within a repo

2 Triggers an execution of the pipeline when a new version is found, builds a new image with build ID

3 Pushes the newly built image tagged with build ID to ECR repo

4 Invokes a Lambda function to trigger application deployment

5 Leverages Kubernetes Python SDK to update a deployment

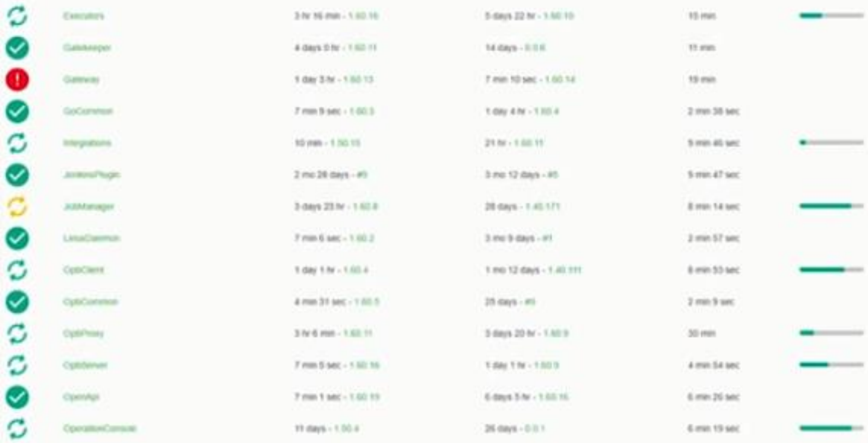
6 Fetches new container image and performs a rolling update of deployment

CI/CD at Microfocus on Kubernetes

SRF CI/CD with Kubernetes

Around 100 developers total

- 13 Dev Teams writing Java, NodeJS & Go
- Commits trigger auto deployment
- ~40 different git repos



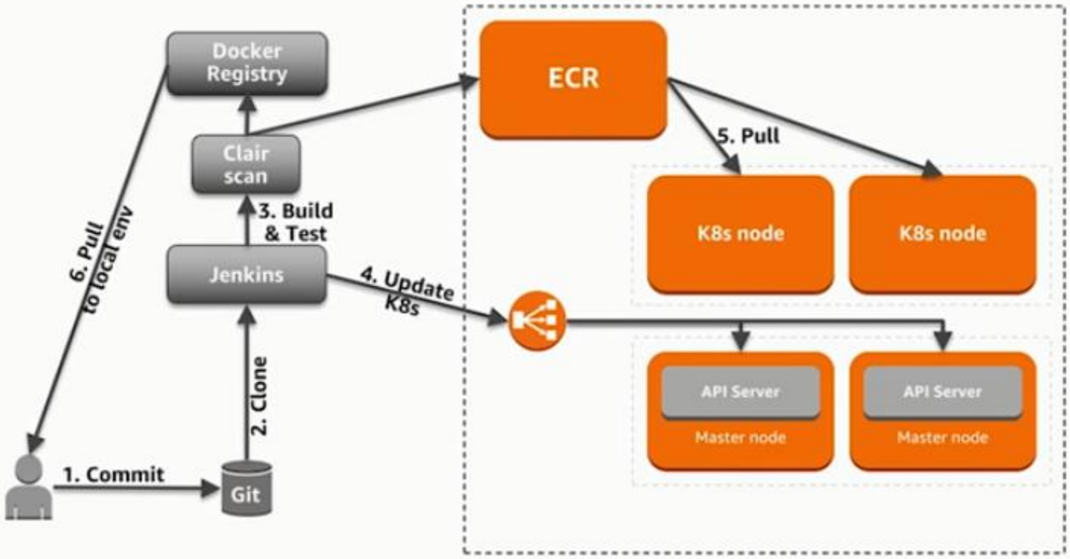
⟳	Executors	3 hr 16 min - 1.60.16	5 days 22 hr - 1.60.10	15 min
✓	Gatekeeper	4 days 0 hr - 1.60.11	14 days - 0.0.6	11 min
!	Gateway	1 day 3 hr - 1.60.13	7 min 10 sec - 1.60.14	19 min
✓	GoCommon	7 min 9 sec - 1.60.3	1 day 4 hr - 1.60.4	2 min 38 sec
⟳	Intergations	10 min - 1.50.15	21 hr - 1.60.11	5 min 45 sec
✓	JenkinsPlugin	2 mo 28 days - #3	3 mo 12 days - #5	5 min 47 sec
⟳	JobManager	3 days 23 hr - 1.60.8	28 days - 1.45.171	8 min 14 sec
✓	LinuxDaemon	7 min 6 sec - 1.60.2	3 mo 9 days - #1	2 min 57 sec
⟳	OptiClient	1 day 1 hr - 1.60.4	1 mo 12 days - 1.40.111	8 min 53 sec
✓	OptiCommon	4 min 31 sec - 1.60.5	25 days - #3	2 min 9 sec
⟳	OptiProxy	3 hr 6 min - 1.60.11	3 days 20 hr - 1.60.9	30 min
⟳	OptiServer	7 min 5 sec - 1.50.16	1 day 1 hr - 1.60.9	4 min 54 sec
✓	OpenApi	7 min 1 sec - 1.60.15	6 days 5 hr - 1.60.16	6 min 26 sec
⟳	OperationConsole	11 days - 1.50.4	26 days - 0.0.1	6 min 19 sec

SRF CI/CD with Kubernetes

A dedicated DevOps team

- Research into kubernetes & AWS features
- Upgrading to latest kubernetes version
- Implementing Jenkins pipelines
- Tweaking service auto-scaling & resource limits
- Improving Kubernetes security
- Deploying SRF to production

DevOps



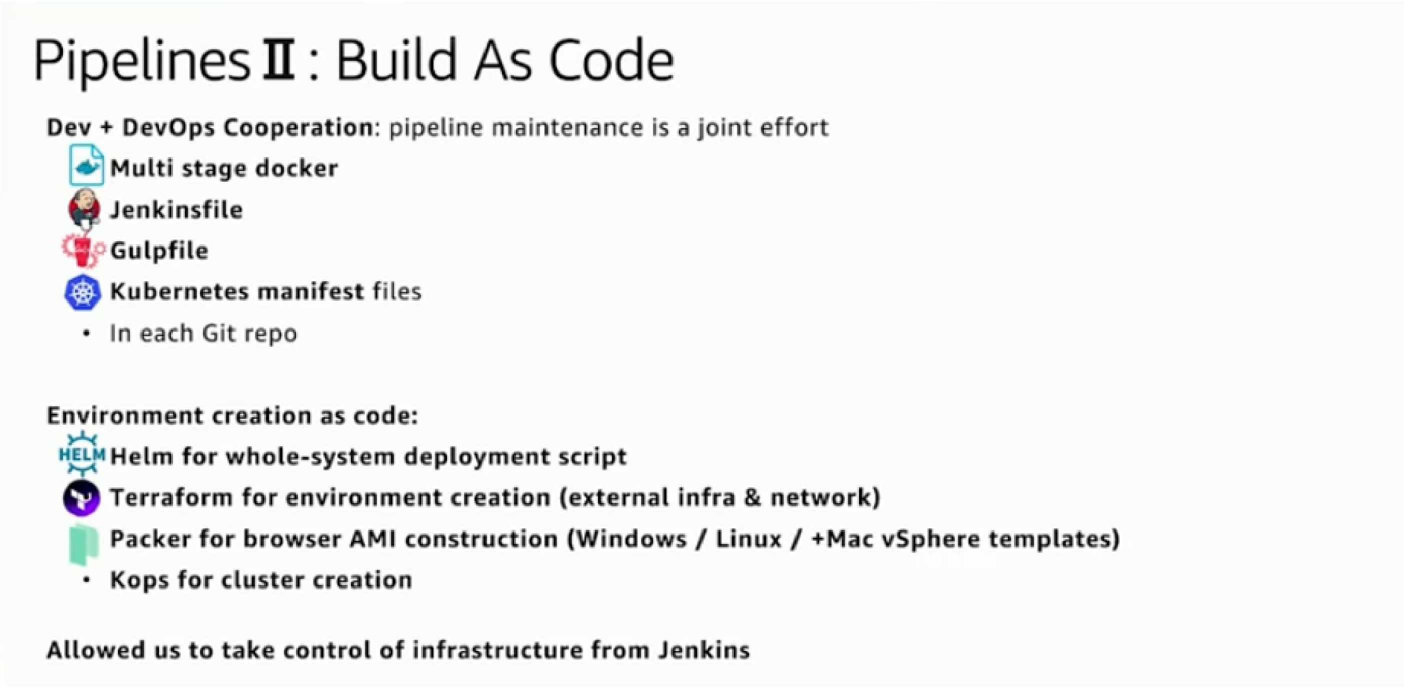
Pipelines I : Jenkins jobs

Parametrized Infra Jenkins Jobs

- Commit, Compile Docker Build, Deploy, Test, etc.
- Manager Jobs for each service
- A specialized build for:
  - Common modules
  - AMI Creation

Pipelines I : Issues with Jenkins jobs

- **Hard to trace:** Multiple cascading jobs with parameters
- **Dev needs DevOps:** for build changes
- **Slow pipeline creation**: hard to create new build processes (next versions / private branches)
- **No build versioning:** build structure is managed manually, backed up as a whole



Pipelines Ⅱ : Build As Code

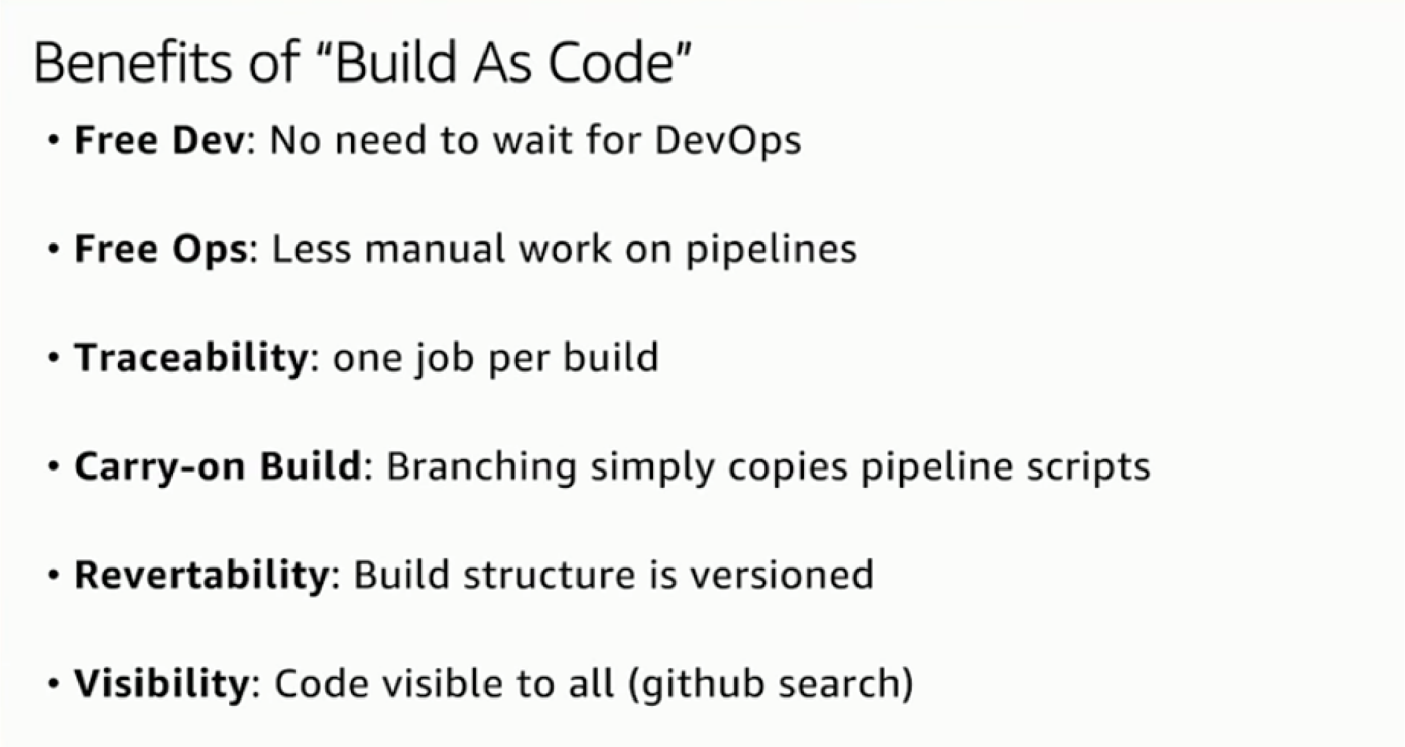
Dev + DevOps Cooperation: pipeline maintenance is a joint effort

- **Multi stage docker**
- **Jenkinsfile**
- **Gulpfile**
- **Kubernetes manifest** files
  - In each Git repo

Environment creation as code:

- **Helm for whole-system deployment script**
- **Terraform for environment creation (external infra & network)**
- **Packer for browser AMI construction (Windows / Linux / +Mac vSphere templates)**
  - **Kops for cluster creation**

Allowed us to take control of infrastructure from Jenkins

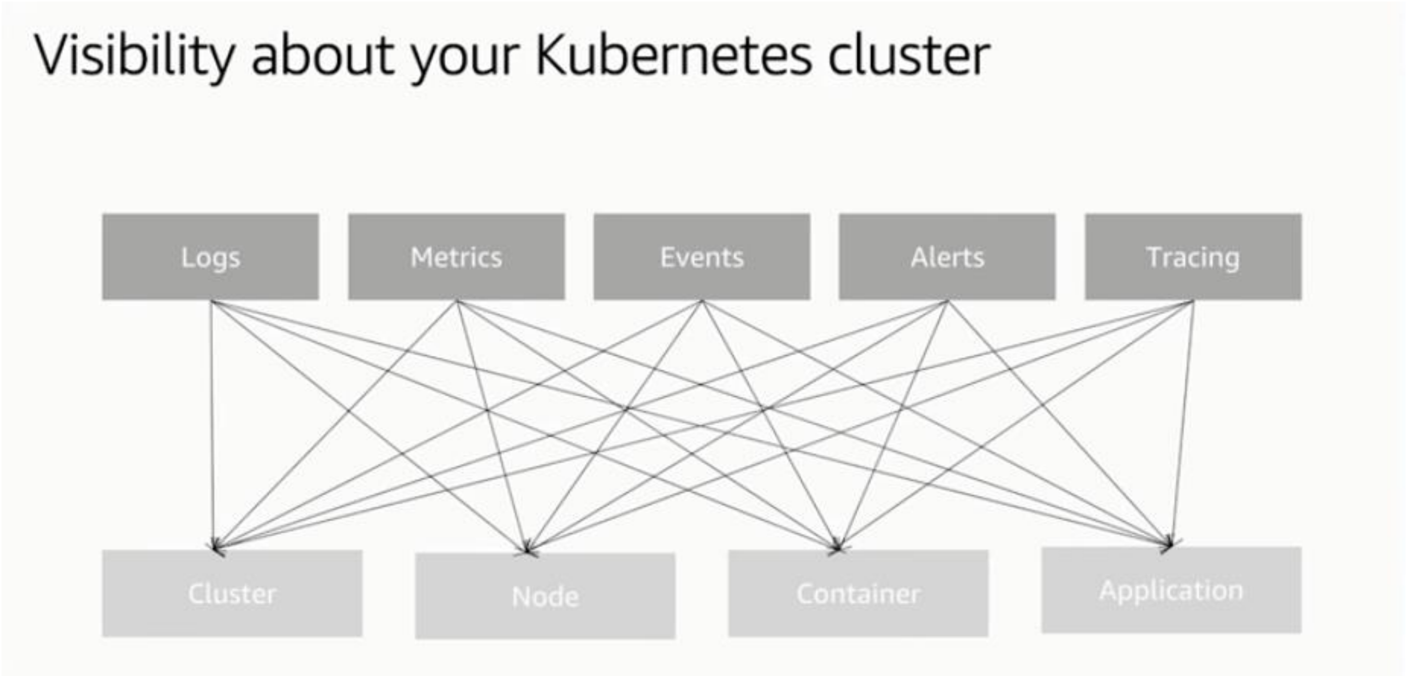


Benefits of "Build As Code"

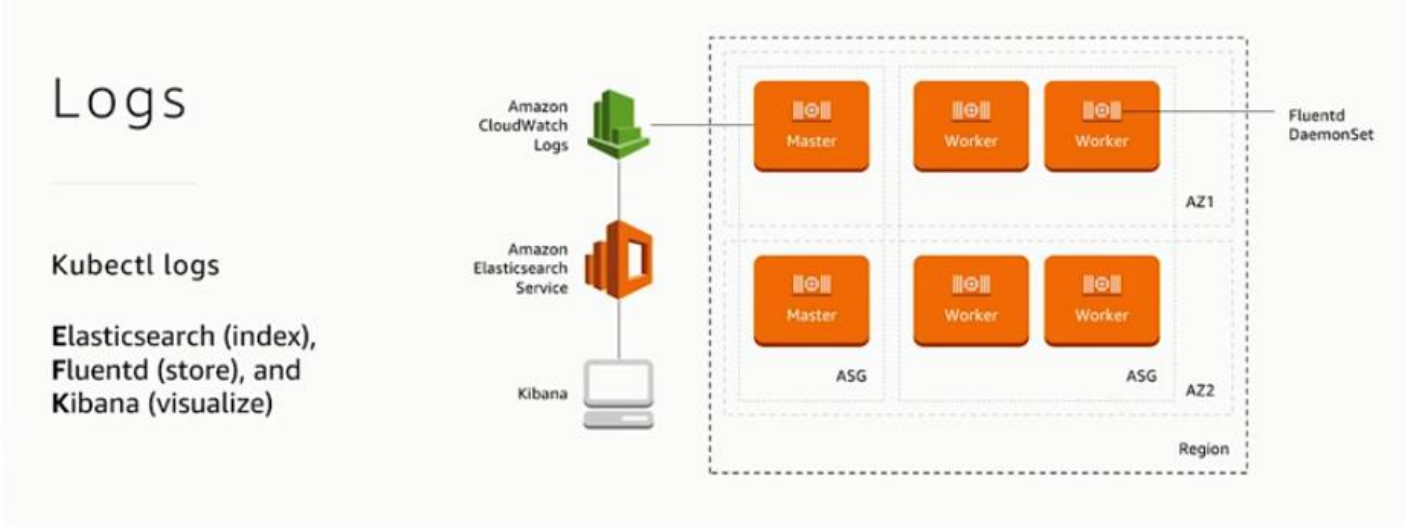
- **Free Dev**: No need to wait for DevOps
- **Free Ops**: Less manual work on pipelines
- **Traceability**: one job per build
- **Carry-on Build**: Branching simply copies pipeline scripts
- **Revertability**: Build structure is versioned
- **Visibility**: Code visible to all (github search)

3

Visibility in cluster



Visibility about your Kubernetes cluster

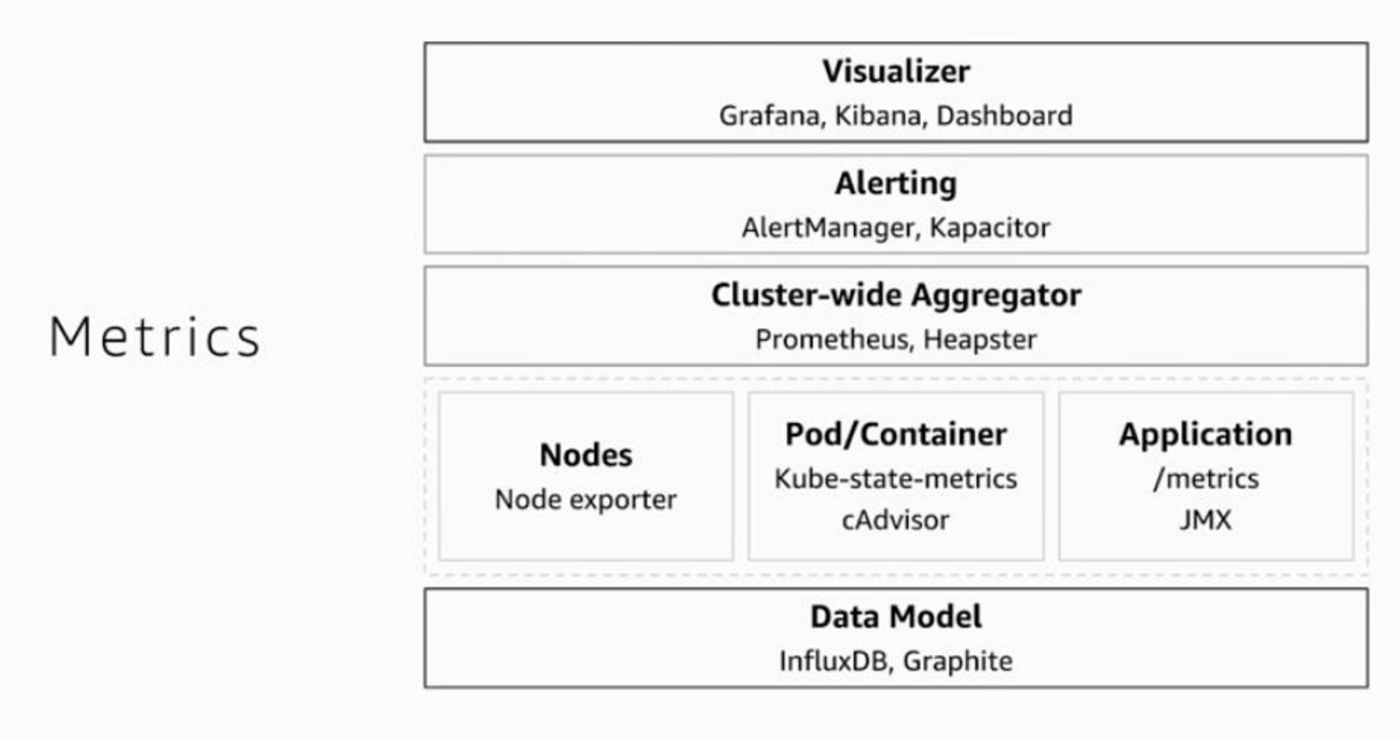


Logs

Kubectl logs

Elasticsearch (index), **F**luentd (store), and **K**ibana (visualize)

Fluentd DaemonSets scrapes the logs and sends them to CloudWatch Logs, you can then use a plugin from CloudWatch Logs to connect to Elasticsearch where you can set up your Kibana dashboards.



Metrics

Visualizer
Grafana, Kibana, Dashboard

Alerting
AlertManager, Kapacitor

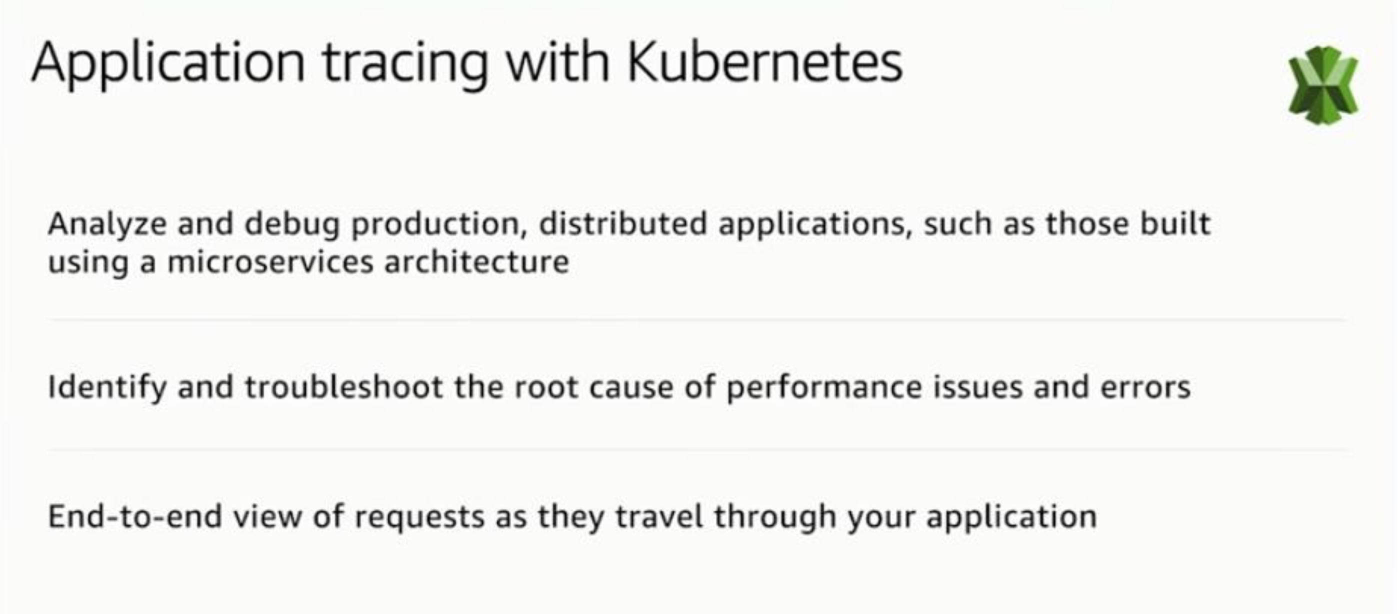
Cluster-wide Aggregator
Prometheus, Heapster

Nodes
Node exporter

Pod/Container
Kube-state-metrics
cAdvisor

Application
/metrics
JMX

Data Model
InfluxDB, Graphite

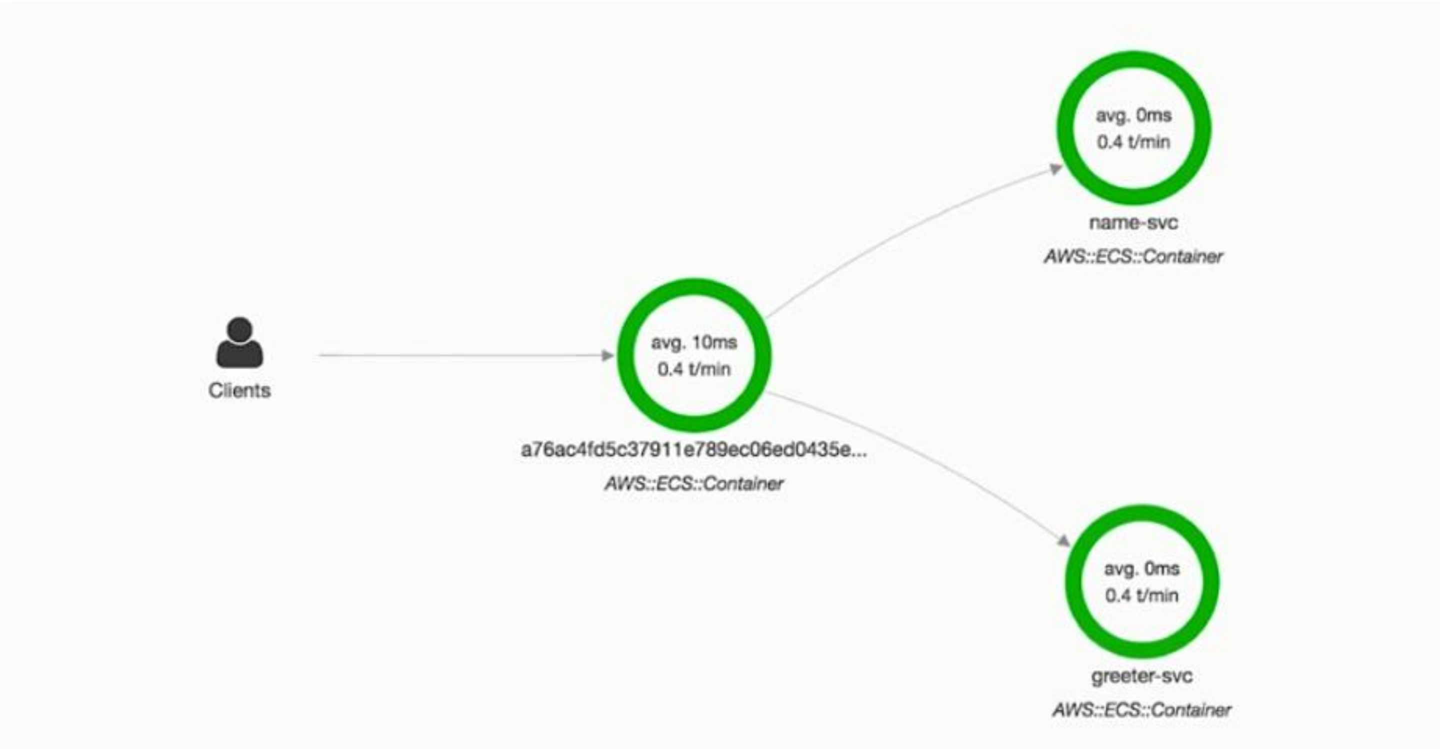


Application tracing with Kubernetes

Analyze and debug production, distributed applications, such as those built using a microservices architecture

Identify and troubleshoot the root cause of performance issues and errors

End-to-end view of requests as they travel through your application



You can instrument with the XRay SDK that will use the XRay daemon and add to your nodes to collect traces

Visibility in cluster at Microfocus

Logging with Kubernetes

Centralized logging solution:

Elastic / FluentD / Kibana

Pods log to stdout

FluentD DaemonSet:

Sends logs to centralized ElasticSearch

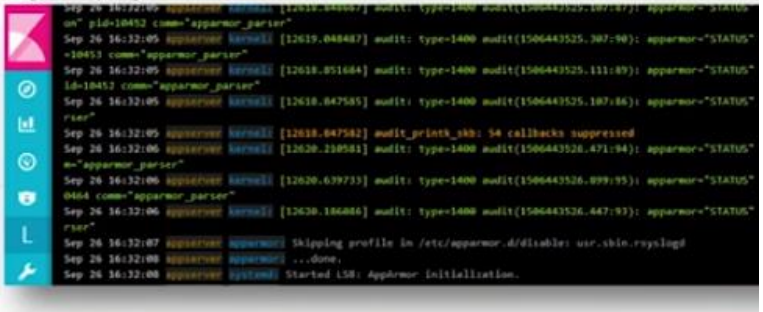
External logs:

- Browser Machine Logs
- Script Execution Logs

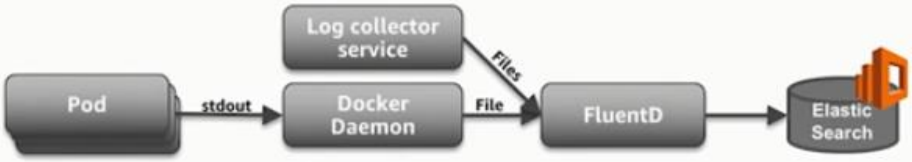
Collected to FluentD Directory (applicative)

Collector uses FluentD on its machine

Logtrail plugin in kibana:



```
Sep 26 16:32:05 appserver kernel: [12618.848667] audit: type=1400 audit(1506443525.107:87): apparmor="STATUS"
on" pid=10452 comm="apparmor_parser"
Sep 26 16:32:05 appserver kernel: [12619.048487] audit: type=1400 audit(1506443525.307:90): apparmor="STATUS"
=10453 comm="apparmor_parser"
Sep 26 16:32:05 appserver kernel: [12618.851684] audit: type=1400 audit(1506443525.111:89): apparmor="STATUS"
id=10452 comm="apparmor_parser"
Sep 26 16:32:05 appserver kernel: [12618.847585] audit: type=1400 audit(1506443525.107:86): apparmor="STATUS"
rser"
Sep 26 16:32:05 appserver kernel: [12618.847582] audit_printk_skb: 54 callbacks suppressed
Sep 26 16:32:06 appserver kernel: [12620.210581] audit: type=1400 audit(1506443526.471:94): apparmor="STATUS"
m="apparmor_parser"
Sep 26 16:32:06 appserver kernel: [12620.639733] audit: type=1400 audit(1506443526.899:95): apparmor="STATUS"
0464 comm="apparmor_parser"
Sep 26 16:32:06 appserver kernel: [12620.186086] audit: type=1400 audit(1506443526.447:93): apparmor="STATUS"
rser"
Sep 26 16:32:07 appserver apparmor: Skipping profile in /etc/apparmor.d/disable: usr.sbin.rsyslogd
Sep 26 16:32:08 appserver apparmor: ...done.
Sep 26 16:32:08 appserver systemd: Started LSB: AppArmor initialization.
```



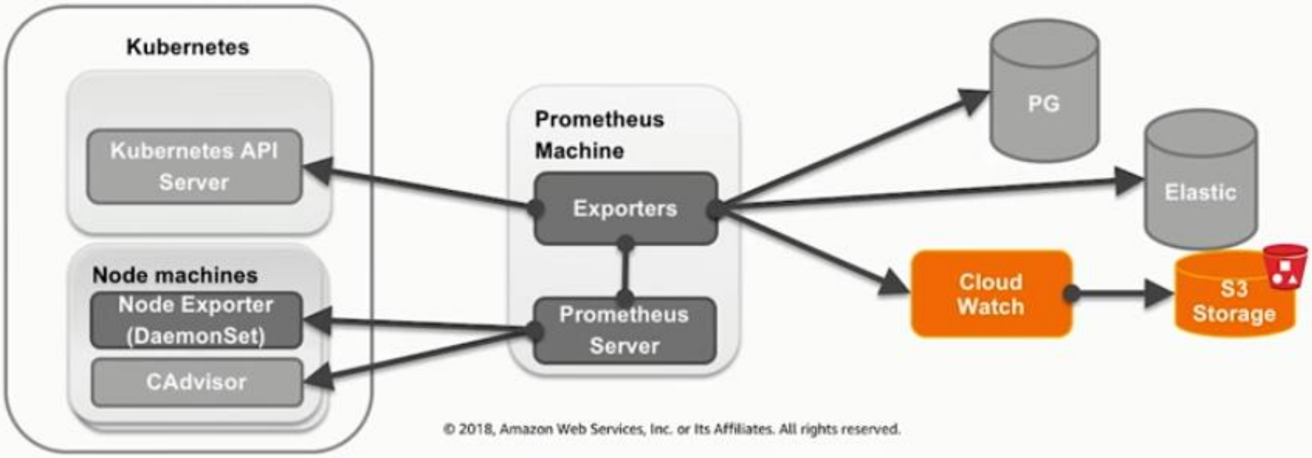
Monitoring Setup

Monitoring/alerting tool: **Prometheus**

Node exporter as DaemonSet for system metrics

External Prom dockers – outside of k8s cluster

cAdvisor collects pods metrics



Application Monitoring & Alerts

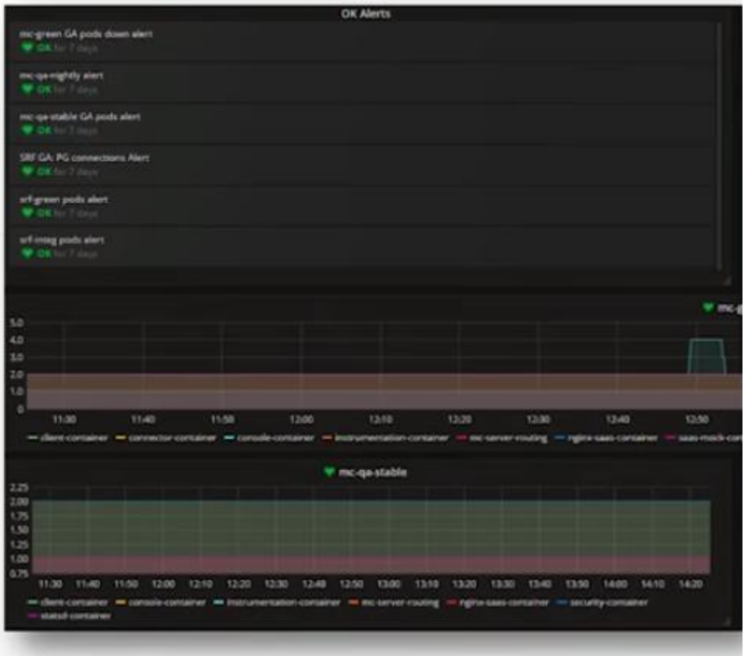
Sytem Metrics tracked:

- **Liveliness**: Per System & Per service
- **System**: CPU / Memory / IO latency
- **API**: Latency, response rate, errors
- **Custom:** Browser pools & Execution slots

Alerts:

Alerts on system limits

Autoscaling Thresholds at 75% of limits



OK Alerts

mc-green GA pods down alert
OK for 7 days

mc-qa-nightly alert
OK for 7 days

mc-qa-stable GA pods alert
OK for 7 days

SRF GA: PG connections Alert
OK for 7 days

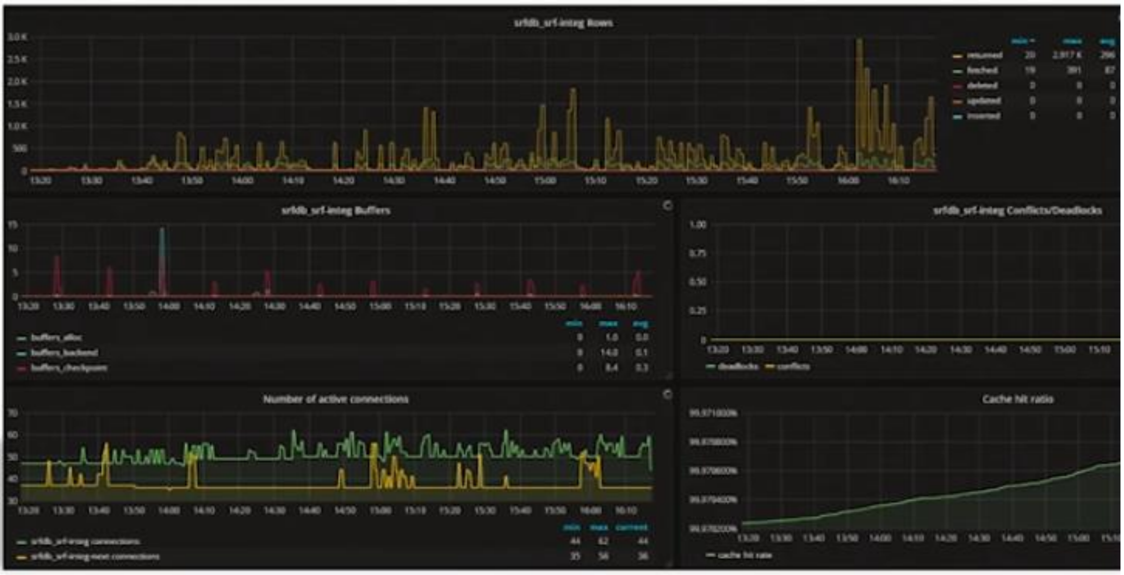
srf-green pods alert
OK for 7 days

srf-integ pods alert
OK for 7 days

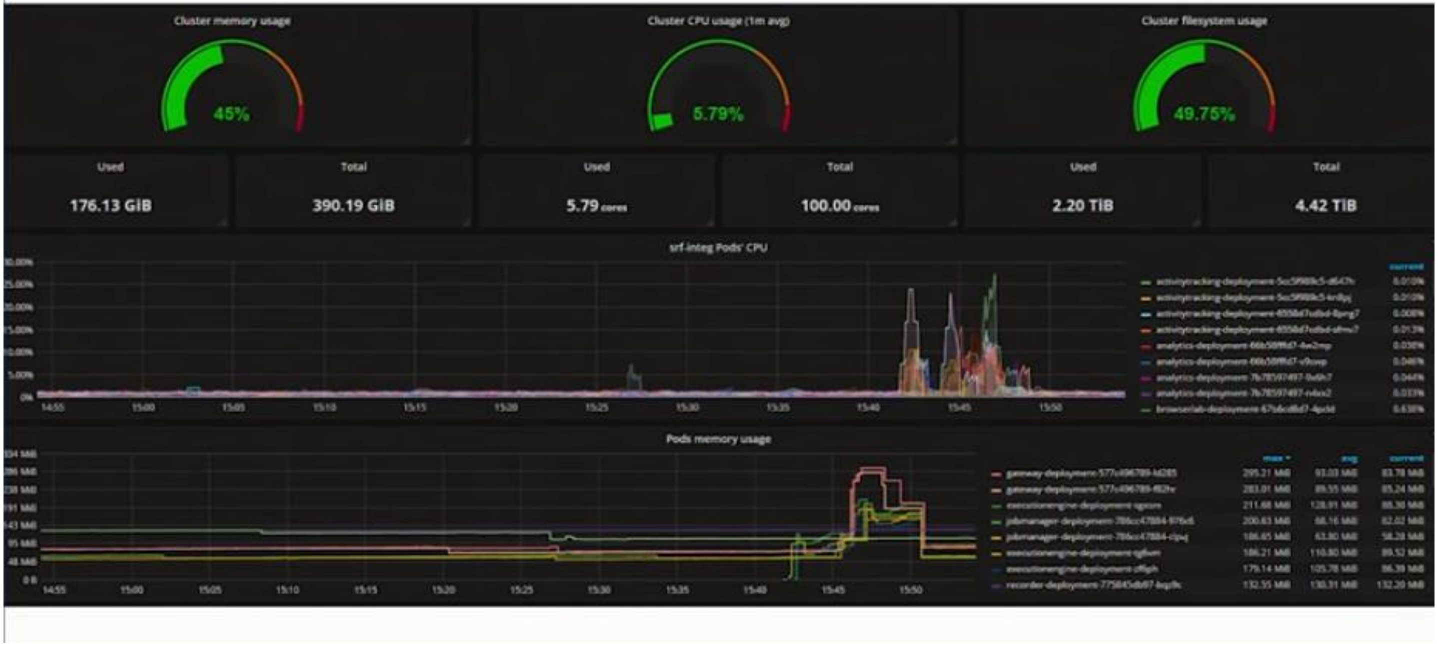
Infra Monitoring

Infra Metrics tracked:

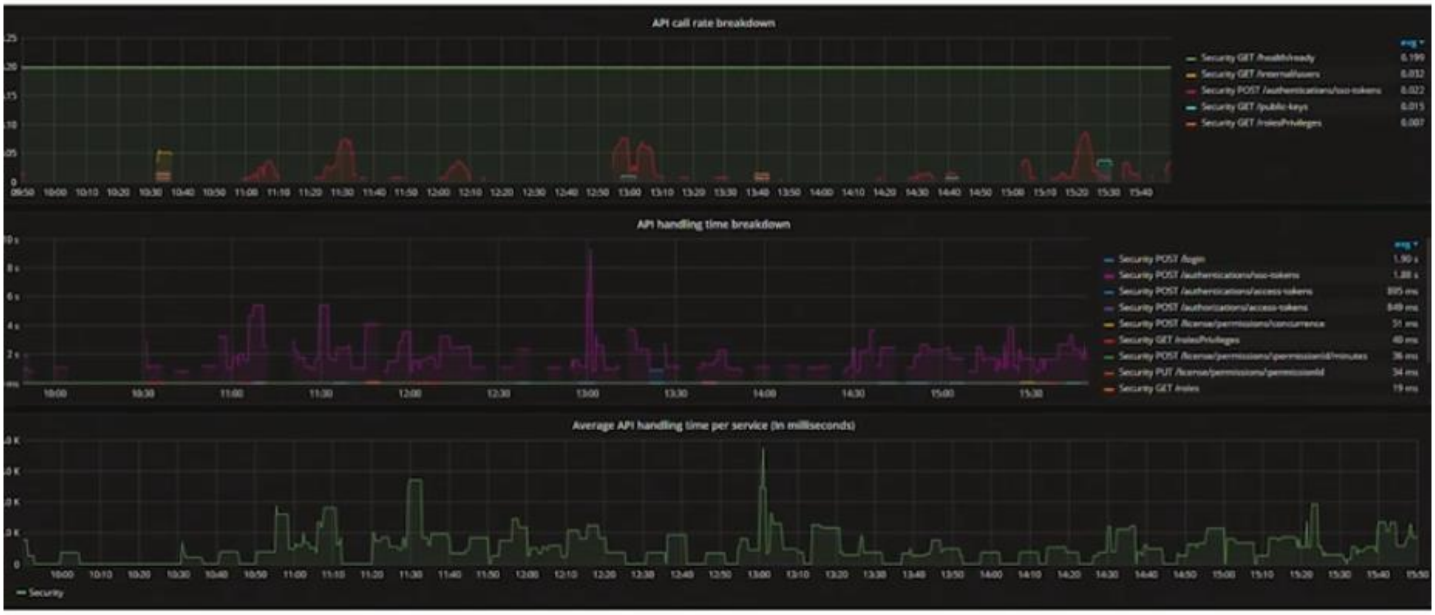
- **Prometheus** (self monitoring)
- **Redis** (Elastic Cache)
- **Postgres** DB
- **ElasticSearch**
- **RabbitMQ**



Grafana – Cluster dashboard



Grafana – API Dashboard


API call rate breakdown
avg
Security GET /health/ready 0.199
Security GET /internal/users 0.032
Security POST /authentications/sso-tokens 0.022
Security GET /public-keys 0.015
Security GET /rolesPrivileges 0.007
API handling time breakdown
avg
Security POST /login 1.90 s
Security POST /authentications/sso-tokens 1.88 s
Security POST /authentications/access-tokens 895 ms
Security POST /authorizations/access-tokens 849 ms
Security POST /license/permissions/concurrence 51 ms
Security GET /rolesPrivileges 40 ms
Security POST /license/permissions/:permissionId/minutes 36 ms
Security PUT /license/permissions/:permissionId 34 ms
Security GET /roles 19 ms
Average API handling time per service (in milliseconds)
Security

References

github.com/aws-samples/aws-workshop-for-kubernetes