CI/CD for Microservices

Goal:

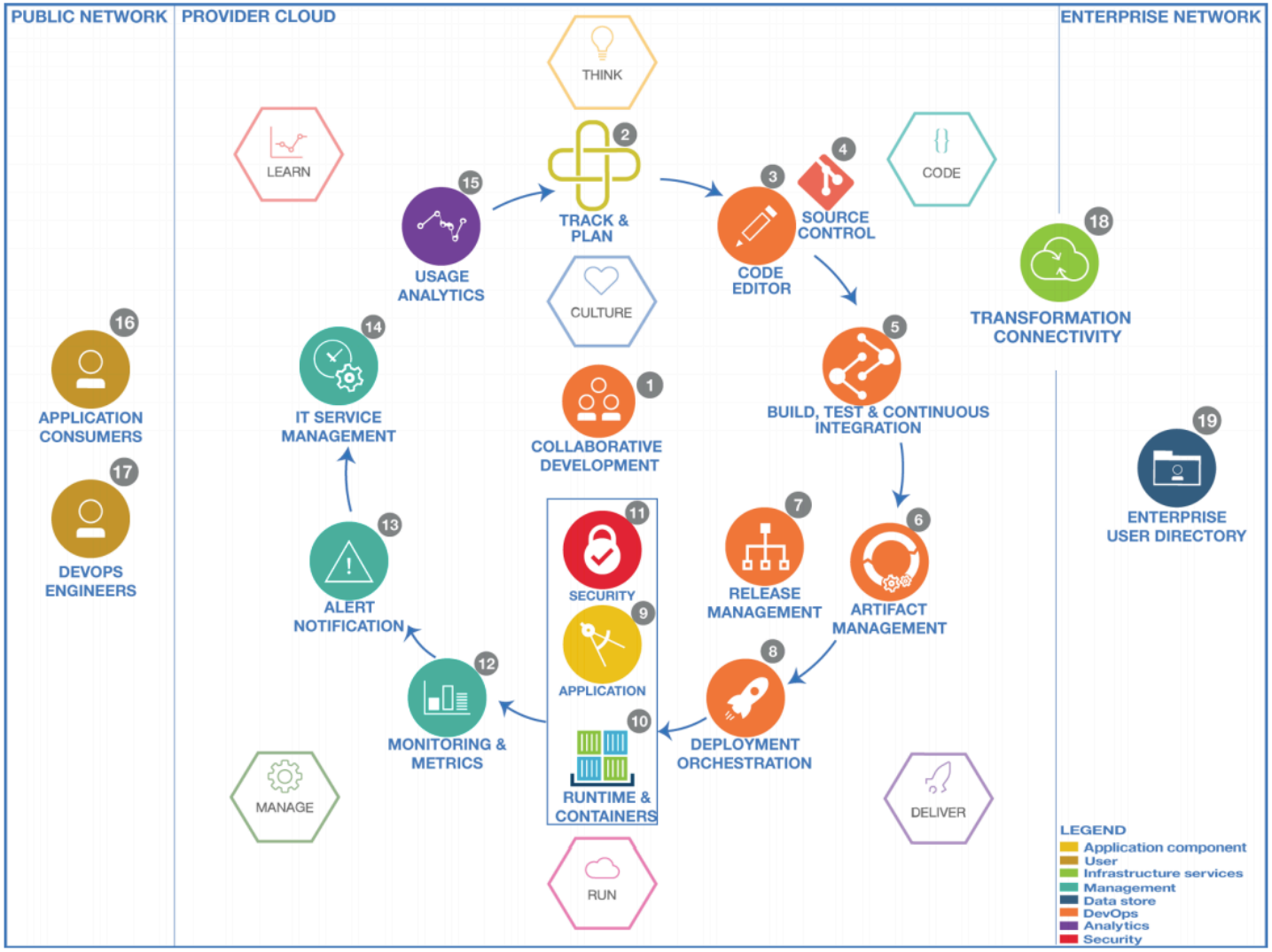
- Continuous Integration and Continuous Deployment
- For Microservices

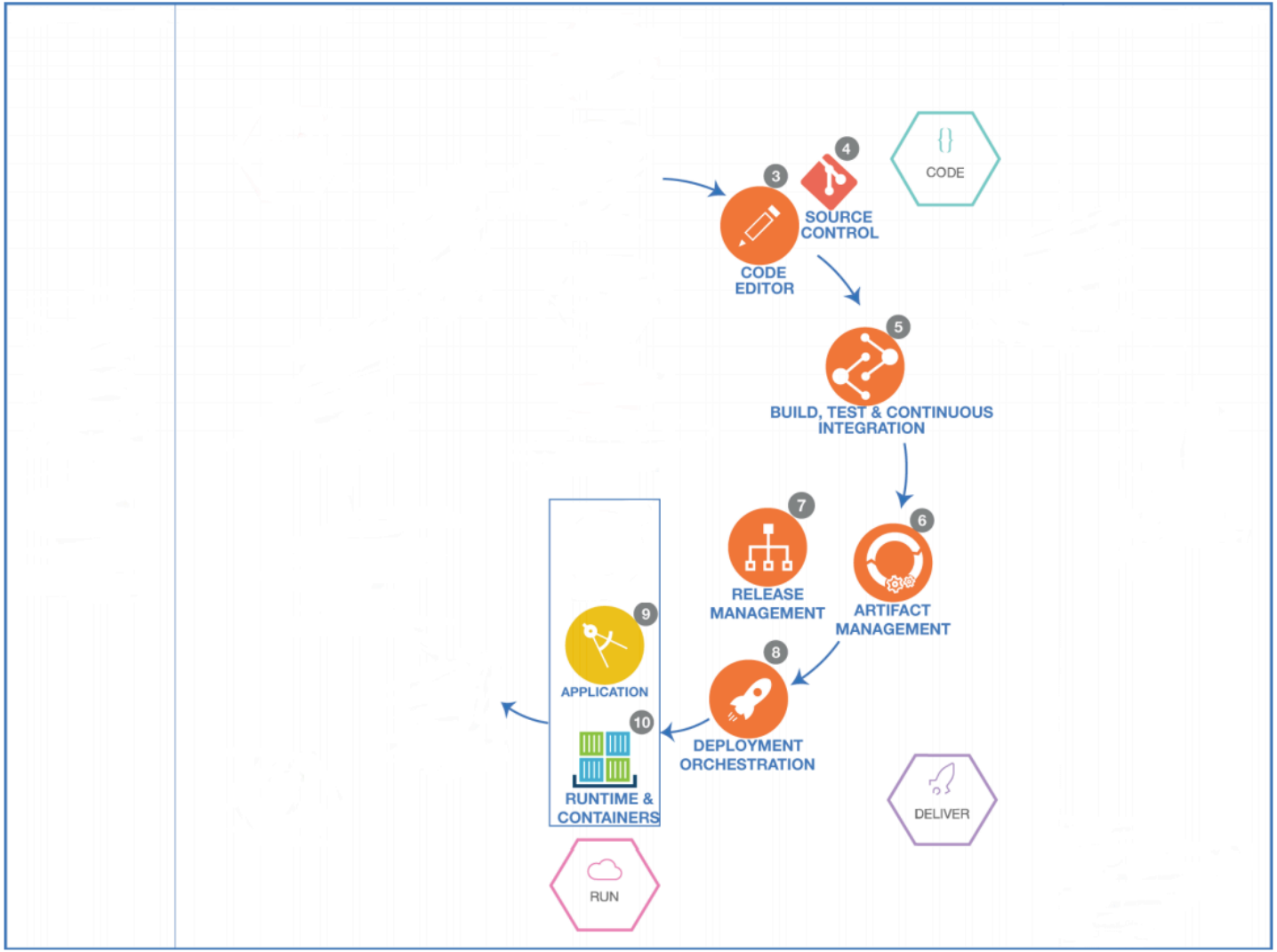
Best practices:

- Release often,
- Blue green deployment,
- Canary deployment,
- A/B Testing,
- Commit daily, reduce branching.

https://12factor.net/

1. Codebase: each codebase has its own CSM repo
2. Dependencies: explicitly declare and isolate dependencies
3. Config: store config in the environment, not in the application
4. Backing services: treat backing services, local or remote, as attached but swappable resources
5. Build, release, run: Strictly separate build and run stages
6. Processes: execute the app as one or more stateless processes running in the environment
7. Port binding: export services via port binding, do not rely on runtime injection
8. Concurrency: scale out via the process model, so that the application is able to span multiple processes on multiple physical machines
9. Disposability: maximize robustness with fast startup and graceful shutdown, elastic scaling, and rapid deployment
10. Dev/prod parity:
    1. Small time gap: from code to deployment asap
    2. Small personnel gap: developer is involved in dev, ops and prod
    3. Small tools gap: keep dev, staging, and prod as similar as possible
11. Logs: Treat logs as event streams, use stdout
12. Admin processes: Run admin/management tasks as one-off processes w the same rules as above


PUBLIC NETWORK
PROVIDER CLOUD
ENTERPRISE NETWORK
THINK
LEARN
CODE
TRACK & PLAN
SOURCE CONTROL
USAGE ANALYTICS
CODE EDITOR
CULTURE
TRANSFORMATION CONNECTIVITY
APPLICATION CONSUMERS
IT SERVICE MANAGEMENT
COLLABORATIVE DEVELOPMENT
BUILD, TEST & CONTINUOUS INTEGRATION
ENTERPRISE USER DIRECTORY
DEVOPS ENGINEERS
SECURITY
RELEASE MANAGEMENT
ARTIFACT MANAGEMENT
ALERT NOTIFICATION
APPLICATION
MONITORING & METRICS
DEPLOYMENT ORCHESTRATION
RUNTIME & CONTAINERS
MANAGE
DELIVER
RUN
LEGEND
Application component
User
Infrastructure services
Management
Data store
DevOps
Analytics
Security


CODE
4
3
SOURCE CONTROL
CODE EDITOR
5
BUILD, TEST & CONTINUOUS INTEGRATION
7
6
RELEASE MANAGEMENT
ARTIFACT MANAGEMENT
9
APPLICATION
8
10
DEPLOYMENT ORCHESTRATION
RUNTIME & CONTAINERS
DELIVER
RUN

CNCF Cloud Native Landscape
2019-10-11T06:02:38Z e06bcda

Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io

Greyed logos are not open source

Database | Streaming & Messaging | Application Definition & Image Build | Continuous Integration & Delivery

App Definition and Development

Scheduling & Orchestration | Coordination & Service Discovery | Remote Procedure Call | Service Proxy | API Gateway | Service Mesh

Orchestration & Management

Cloud Native Storage | Container Runtime | Cloud Native Network

Runtime

Automation & Configuration | Container Registry | Security & Compliance | Key Management

Provisioning

Kubernetes Certified Service Provider

Kubernetes Training Partner

Special

Platform

Certified Kubernetes - Distribution

Certified Kubernetes - Hosted

Certified Kubernetes - Installer

PaaS/Container Service

Observability and Analysis

Monitoring

Logging

Tracing

Chaos Engineering

Serverless

Members

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

l.cncf.io

CLOUD NATIVE COMPUTING FOUNDATION

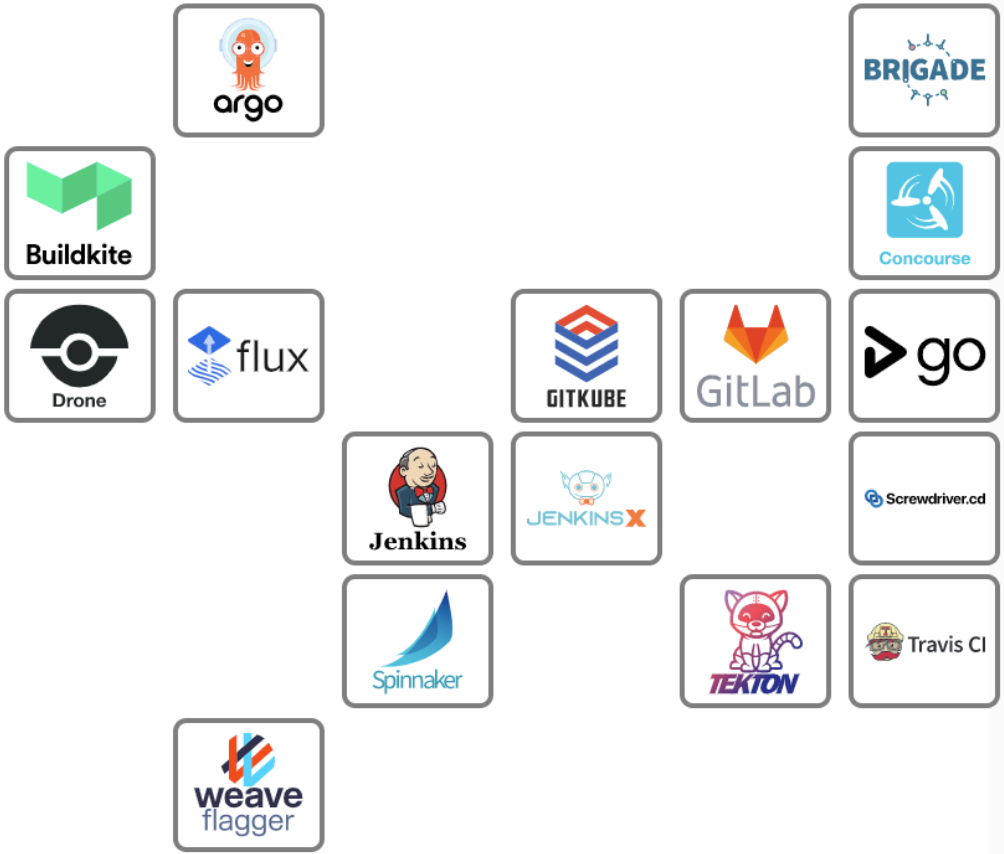
CLOUD NATIVE Landscape

Redpoint | Amplify

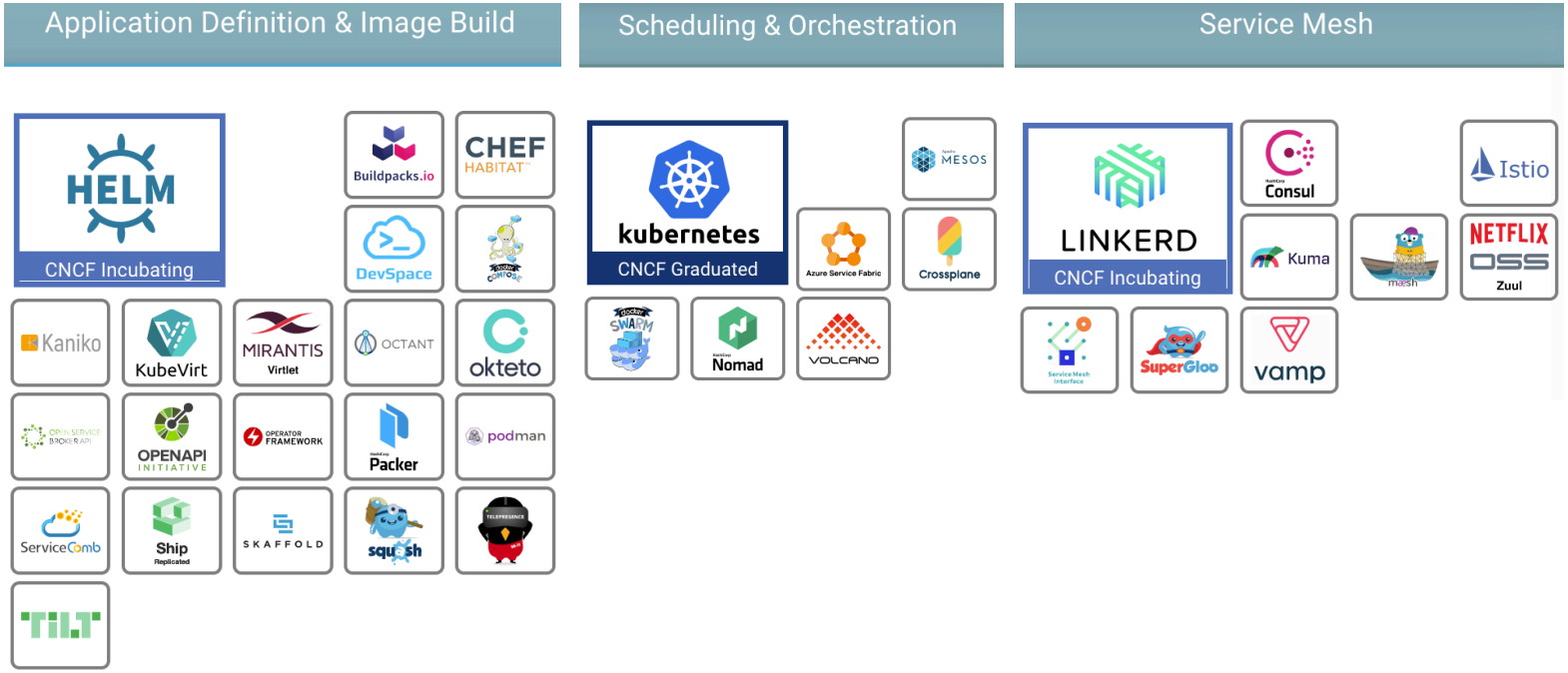
https://landscape.cncf.io/license=open-source

Continuous Integration & Delivery

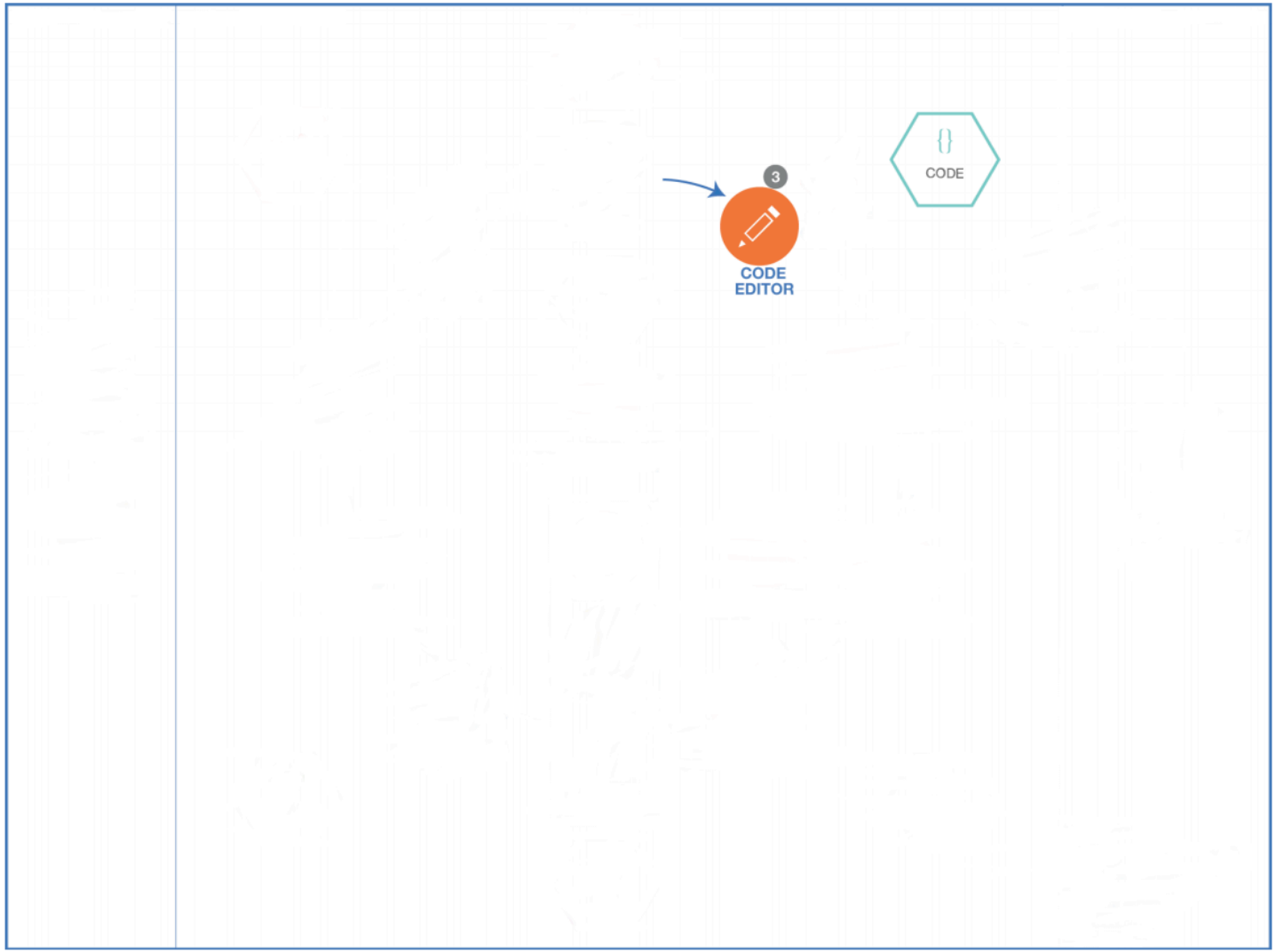
https://landscape.cncf.io/license=open-source



CNCF.io


Application Definition & Image Build
Scheduling & Orchestration
Service Mesh
HELM
CNCF Incubating
Buildpacks.io
CHEF HABITAT
DevSpace
Kaniko
KubeVirt
MIRANTIS Virtlet
OCTANT
okteto
OPENAPI INITIATIVE
OPERATOR FRAMEWORK
Packer
podman
ServiceComb
Ship Replicated
SKAFFOLD
squash
TILT
kubernetes
CNCF Graduated
MESOS
Azure Service Fabric
Crossplane
Nomad
VOLCANO
LINKERD
CNCF Incubating
Consul
Kuma
Istio
NETFLIX OSS Zuul
SuperGloo
vamp

CI


{}
CODE
3
CODE
EDITOR

Build, Test, Run

Apache Maven

```
$ brew install maven
$ mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes
-DgroupId=com.remkohdev.app -DartifactId=my-app
$ cd my-app
$ mvn clean install
$ mvn test
$ java -cp target/my-app-1.0-SNAPSHOT.jar com.remkohdev.app.App
```

Spring Framework

```
$ brew tap pivotal/tap
$ brew install springboot
$ spring init --dependencies=web,data-rest,thymeleaf MyApp
$ cd MyApp
$ mvn clean install
$ mvn test
$ mvn spring-boot:run
```

Node.js - Loopback

```
$ npm install -g loopback-cli
$ lb
? What's the name of your application? MyNodeApp
? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind? api-server
$ cd MyNodeApp
$ npm install
$ npm test
$ npm start
```

Test

Spring JUnit

```
@RunWith(SpringRunner.class)
@WebMvcTest(value = APIController.class, secure = false)
public class APIControllerTest {
   @Autowired
   private MockMvc mockMvc;
   @Test
   public void getMessage() throws Exception {

      String name = "venus";
      RequestBuilder requestBuilder = MockMvcRequestBuilders
            .get("/api/hello?name="+name)
            .accept(MediaType.APPLICATION_JSON);
      MvcResult result = (MvcResult) mockMvc.perform(requestBuilder)
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.message", is("Hello "+name)))
            .andDo(print())
            .andReturn();
      System.out.println(result.getResponse());
   }
}
```

Test

Mochajs

```
let request = require( 'supertest' )
let assert = require( 'chai' ).assert
let serverPromise = require( '../server/server' )

describe( 'CRUD', () => {
  let server = null
  before( async function ( ) {
    server = await serverPromise
  }
  it( 'GET /api/hello', function ( done ) {
       request( server ).get( '/v1.1/banks' )
          .set( 'Content-type', 'application/json' )
          .expect( 200 )
          .then( response => {
             assert.isOk( response.body.length > 0 )
             done()
          })
  })
})
```

Test

Postman

```
$ curl -X GET 'http://localhost:8080/api/hello?name=pluto'
```

```
pass = true;
fail = false;
try {
   resp = JSON.parse(pm.response.text());
   pm.test("Check response to have status 200", function () {
      pm.response.to.have.status(200);
   });
   pm.test("Check response includes message", function () {
      pm.expect(pm.response.text()).to.include("message");
   });
   pm.globals.set("env_message", resp["message"]);
}
catch(err) {
   tests["Parse response JSON or tests initialisation"] = fail;
}
```

```
$ newman run postman/springclient-test.postman_collection.json
```

Run

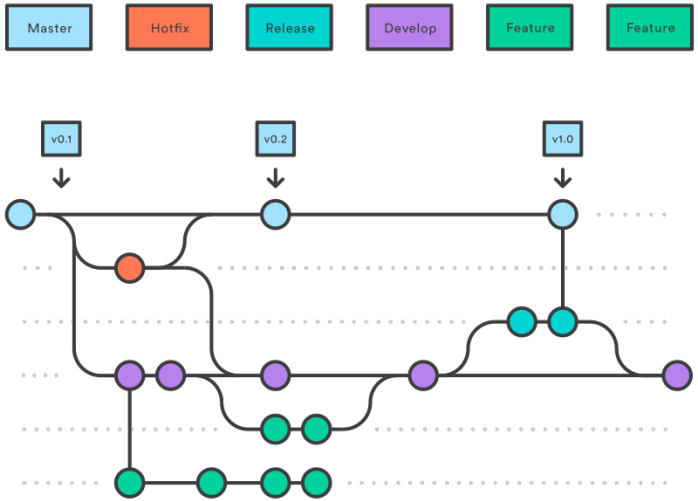
Apache Maven	$ mvn clean install $ java -cp target/my-app-1.0-SNAPSHOT.jar com.remkohdev.app.App
Spring Framework	$ mvn clean install $ mvn spring-boot:run
Node.js - Loopback	$ npm install $ npm start

SCM

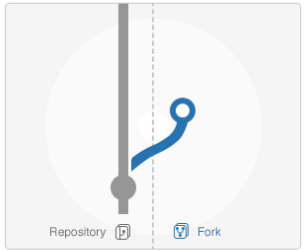
Git

```
$ git add .
$ git commit –m ’my update for my issue’
$ git push
```

Feature branches

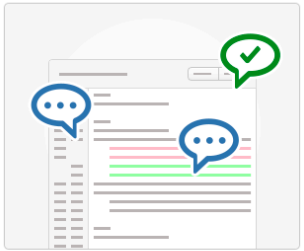


Forks



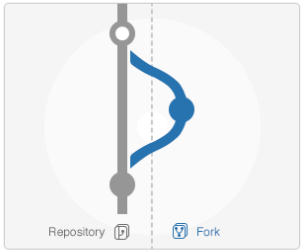
Fork

Develop features on a branch and create a pull request to get changes reviewed.



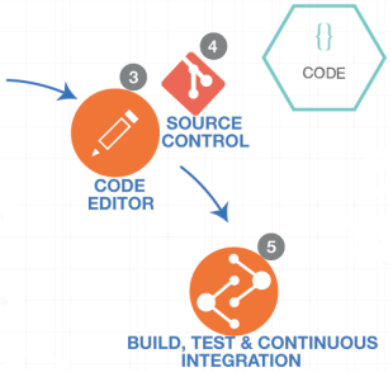
Discuss

Discuss and approve code changes related to the pull request.

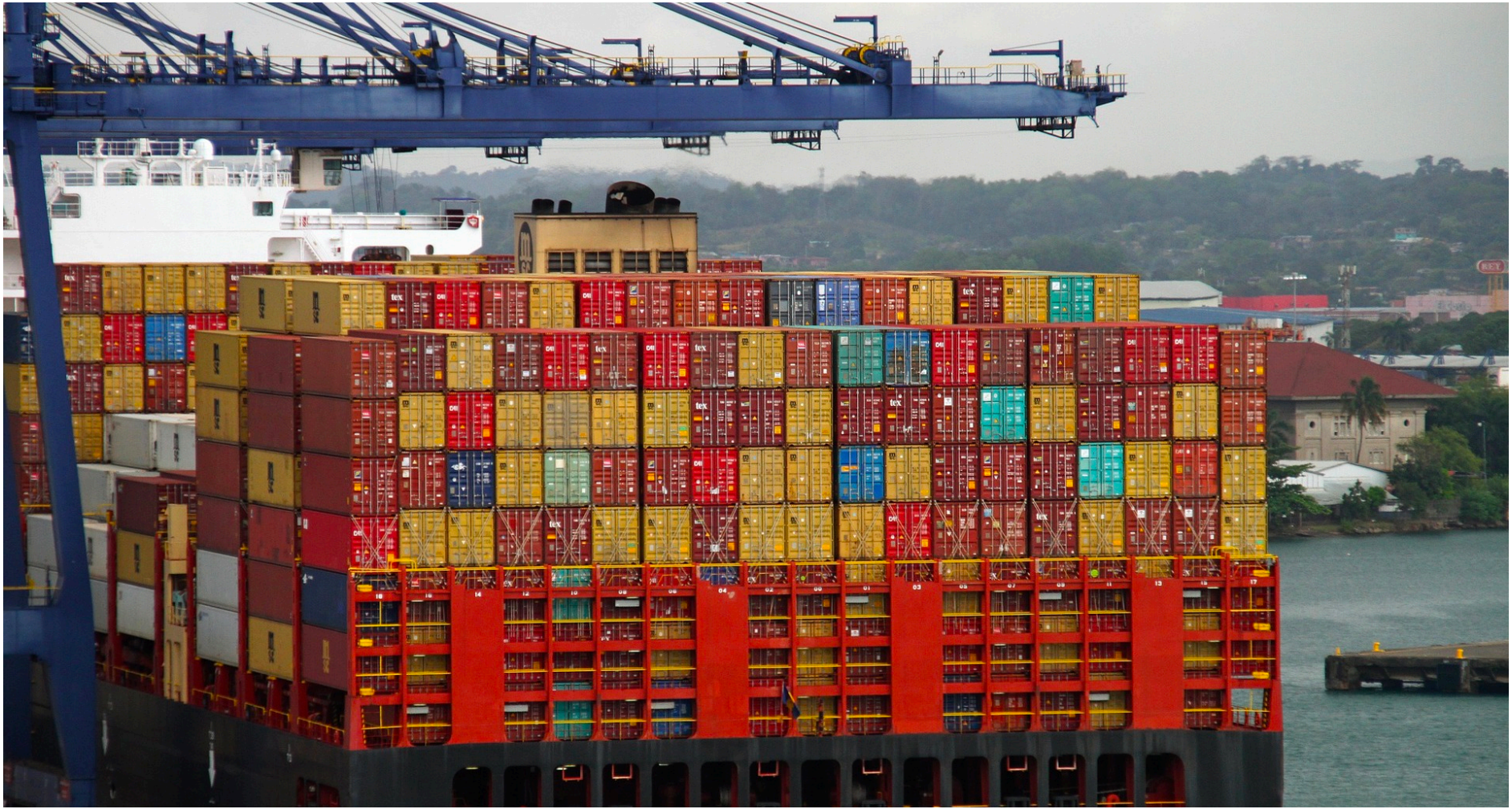


Merge

Merge the branch with the click of a button.


{}
CODE
4
3
SOURCE
CONTROL
CODE
EDITOR
5
BUILD, TEST & CONTINUOUS
INTEGRATION

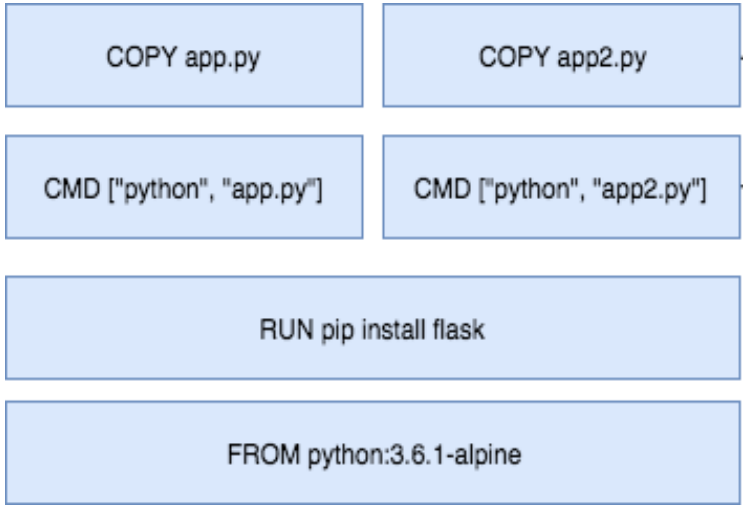
Docker



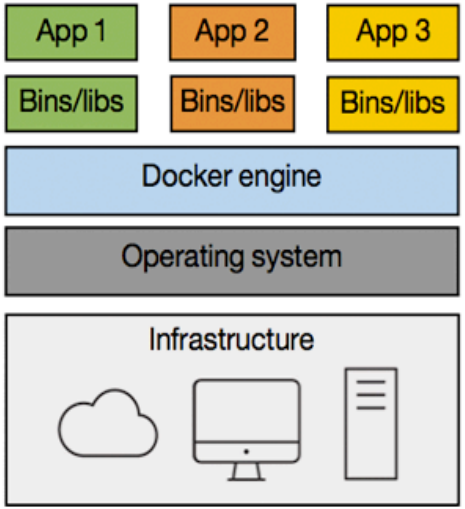
Dockerfile

```
FROM python:3.6.1-alpine
RUN pip install flask
CMD ["python","app2.py"]
COPY app2.py /app2.py
```

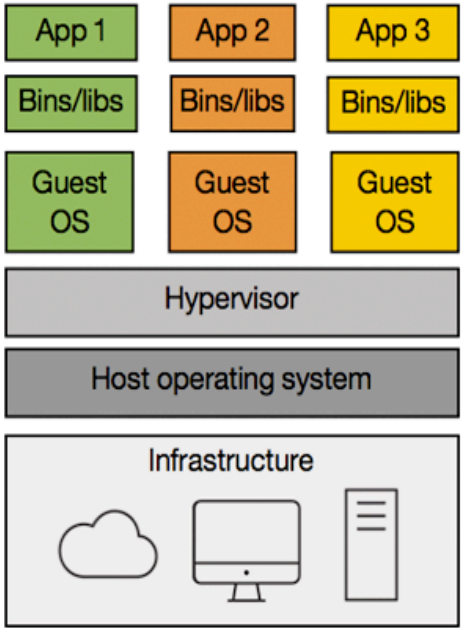
Image Layers



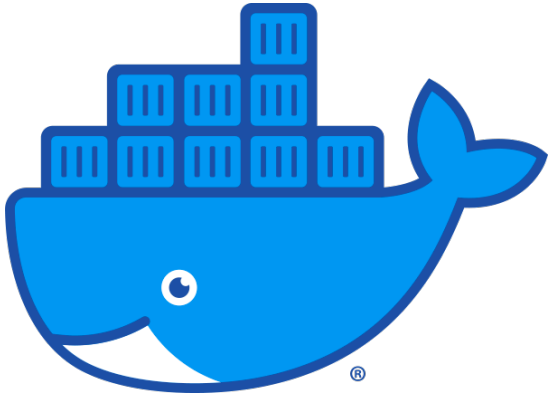
Deployment through Containers



Containers



Virtual Machines



Dockerfile

```
FROM node:11.10.1-stretch-slim

ARG NPM_SCOPE
ARG NPM_REGISTRY_URL
ARG ART_PASSWORD
ARG ART_USERNAME

# Create app directory
WORKDIR /usr/src/app

# Bundle app source
COPY ./package*.json ./
COPY ./npm-shrinkwrap.json ./
# access private npm registry
RUN echo '@'$NPM_SCOPE':registry=https://'$NPM_REGISTRY_URL'\n//
'$NPM_REGISTRY_URL':_password='$ART_PASSWORD'\n//
'$NPM_REGISTRY_URL':username='$ART_USERNAME'\n//
'$NPM_REGISTRY_URL':email='$ART_USERNAME'\n//
'$NPM_REGISTRY_URL':always-auth=true'>.npmrc

RUN npm install
RUN npm audit fix

RUN rm -f .npmrc

COPY . .

EXPOSE 3000

CMD [ "npm", "start" ]
```

docker-compose.yml

```
version: '3'
services:
  my-api:
    build:
      context: .
      dockerfile: ./Dockerfile
      args:
        NPM_SCOPE: '${NPM_SCOPE}'
        NPM_REGISTRY_URL: '${ART_NPM_REGISTRY_URL}'
        ART_PASSWORD: '${ART_BASE64_PASSWORD}'
        ART_USERNAME: '${ART_USERNAME}'
    command: npm start
    ports:
      - ”3000:3000”
    environment:
      NODE_ENV: local
      COUCHDB_USERNAME: ${COUCHDB_USERNAME}
      COUCHDB_PASSWORD: ${COUCHDB_PASSWORD}
      COUCHDB_SERVER: ${COUCHDB_SERVER}
      COUCHDB_PORT: ${COUCHDB_PORT}
      COUCHDB_PROTOCOL: ${COUCHDB_PROTOCOL}
    volumes:
      - .:/usr/src/app

    networks:
     - my-network
```

Docker

```
$ docker build --no-cache -t my-app .
$ docker tag my-app:latest remkohdev/my-app:0.1.0
$ docker run -d --name my-app -p 3000:3000 my-app
$ docker login
$ docker push remkohdev/my-app:0.1.0
```

Docker Compose

```
$ docker-compose build
$ docker-compose up -d
```

CLOUDFOUNDRY

manifest.yml

```
---
applications:
- name: my-app
buildpack: https://github.com/cloudfoundry/nodejs-buildpack
command: node my-app.js
```

```
$ cf push my-app -c "node my-app.js"
```


CODE
4
3
SOURCE CONTROL
CODE EDITOR
5
BUILD, TEST & CONTINUOUS INTEGRATION
7
6
RELEASE MANAGEMENT
ARTIFACT MANAGEMENT
DELIVER

We can build, test and run on localhost

We can collaborate

We can run anywhere

We pushed our build artifact

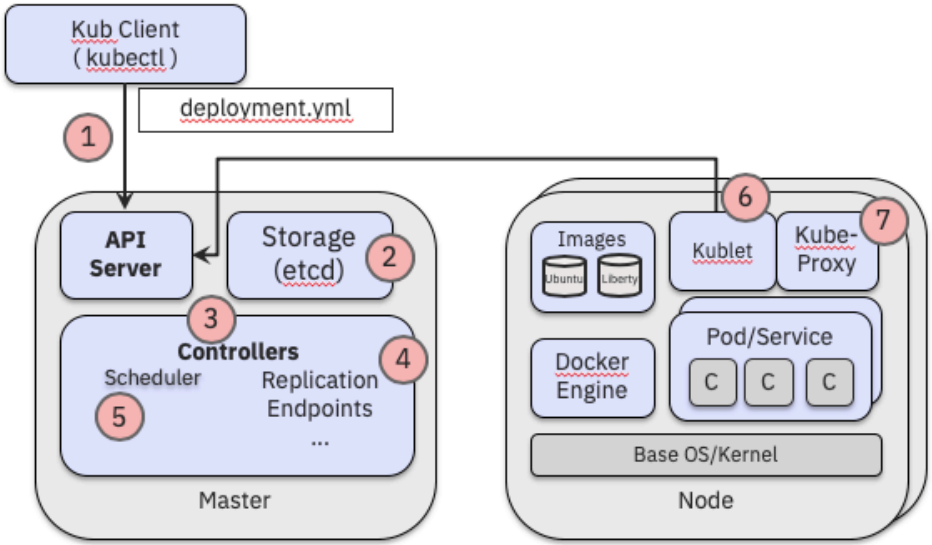
Let’s deploy

Kubernetes

Kubernetes is a container-orchestration system

for automating application deployment, scaling, and management,

originally created by Google, now maintained by the Cloud Native Computing Foundation (CNCF).



1. User via "kubectl" deploys a new application
2. API server receives the request and stores it in the DB (etcd)
3. Watchers/controllers detect the resource changes and act upon it
4. ReplicaSet watcher/controller detects the new app and creates new pods to match the desired # of instances
5. Scheduler assigns new pods to a kubelet
6. Kubelet detects pods and deploys them via the container runing (e.g. Docker)
7. Kubeproxy manages network traffic for the pods – including service discovery and load-balancing

Kubernetes Objects

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.16/

Basic Objects:

- Pod
- Service
- PersistentVolume
- PersistentVolumeClaim
- Namespace
- ConfigMap
- Secret

Higher-Level Abstractions:

- Deployment
- DaemonSet
- StatefulSet
- ReplicaSet
- Job

Custom Resource Definitions (CRD):

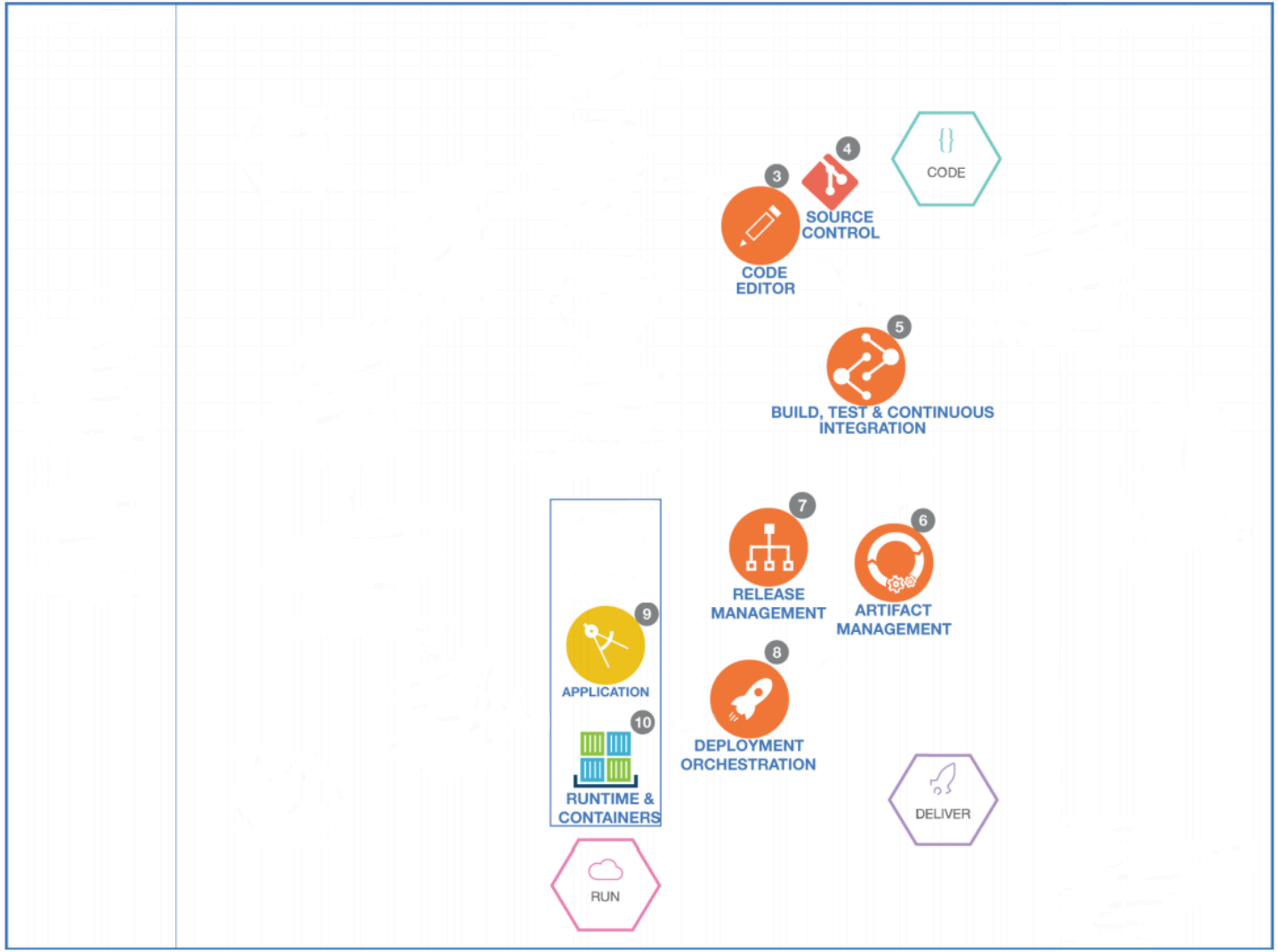
- Operators (https://operatorhub.io/)

deployment.yaml

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: my-app-deployment
  namespace: my-ns
  labels:
    app: my-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: remkohdev/my-app:latest
        ports:
        - name: main
          protocol: TCP
          containerPort: 6661
        envFrom:
        - configMapRef:
            name: my-app-configmap
        resources:
          requests:
            memory: "120M"
            cpu: "500m"
```

Deploy

```
# configmap
kubectl delete configmap -n my-ns my-app-configmap

kubectl create -f ./helm/templates/configmap.yaml

# deployment
kubectl delete deployment -n my-ns my-app-deployment

# while resource still exists wait
rc=$(eval 'kubectl get deployment -n my-ns my-app-deployment')
while [ ! -z "$rc" ]
do
    rc=$(eval 'kubectl get deployment -n my-ns my-app-deployment')
done

kubectl create -f ./helm/templates/deployment.yaml

# service
kubectl delete svc -n my-ns my-app-svc

# while resource still exists wait
rc=$(eval 'kubectl get svc -n my-ns my-app-svc')
while [ ! -z "$rc" ]
do
    rc=$(eval 'kubectl get svc -n my-ns my-app-svc')
done

kubectl create -f ./helm/templates/svc.yaml
```


CODE
4
3
SOURCE CONTROL
CODE EDITOR
5
BUILD, TEST & CONTINUOUS INTEGRATION
7
6
RELEASE MANAGEMENT
ARTIFACT MANAGEMENT
9
APPLICATION
8
DEPLOYMENT ORCHESTRATION
10
RUNTIME & CONTAINERS
DELIVER
RUN

Automation

- Bash
- Docker-compose
- Jenkins

```
docker build --no-cache -t my-app .
docker tag my-app:latest remkohdev/my-app:0.1.0
docker login –u remkohdev
docker push remkohdev/my-app:0.1.0
```

```
kubectl delete configmap -n my-ns my-app-configmap
kubectl create -f ./helm/templates/configmap.yaml

kubectl delete deployment -n my-ns my-app-deployment
rc=$(eval 'kubectl get deployment -n my-ns my-app-deployment')
while [ ! -z "$rc" ]
do
   rc=$(eval 'kubectl get deployment -n my-ns my-app-deployment')
done
kubectl create -f ./helm/templates/deployment.yaml

kubectl delete svc -n my-ns my-app-svc
rc=$(eval 'kubectl get svc -n my-ns my-app-svc')
while [ ! -z "$rc" ]
do
   rc=$(eval 'kubectl get svc -n my-ns my-app-svc')
done
kubectl create -f ./helm/templates/svc.yaml
```

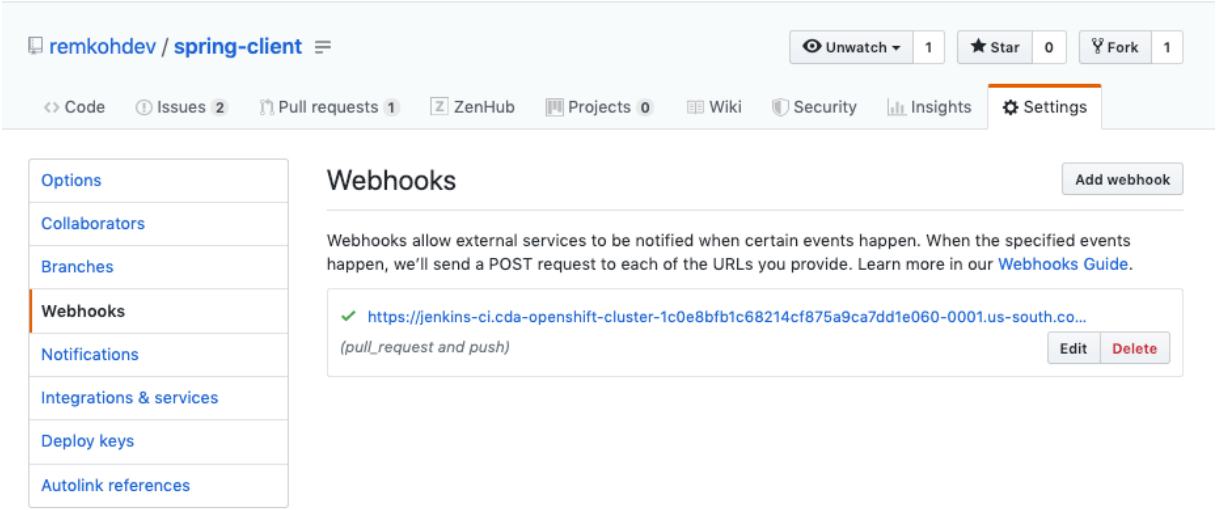
Jenkins

```
node() {

  stage("Checkout SCM") {
    checkout scm;
  }

  stage("Artifactory Configuration") {
    server = Artifactory.server( 'na.artifactory.swg-devops.com' )
    buildInfo = Artifactory.newBuildInfo()
    rtDocker = Artifactory.docker server: server
    env.ART_DOCKER_IMAGE_TAG = "${ART_DOCKER_REPO}/${ART_B
UILD_BRANCH}/${DOCKER_IMAGE_NAME}:${env.ART_DOCKER_TAG}"
  }

  stage("Node Test"){
    sh 'node -v'
  }

  stage("Build Docker Image") {
     docker.build( "${ART_DOCKER_IMAGE_TAG}")
  }

  stage("Push Docker Image") {
     withCredentials([usernamePassword(credentialsId: "$JENKINS_ART
IFACTORY_CREDENTIAL", passwordVariable: 'ART_PW', usernameVaria
ble: 'ART_USER')]) {
     rtDocker.push( "${ART_DOCKER_IMAGE_TAG}",  "${ART_DOCKER_
REPO}" )
     }
    }
```

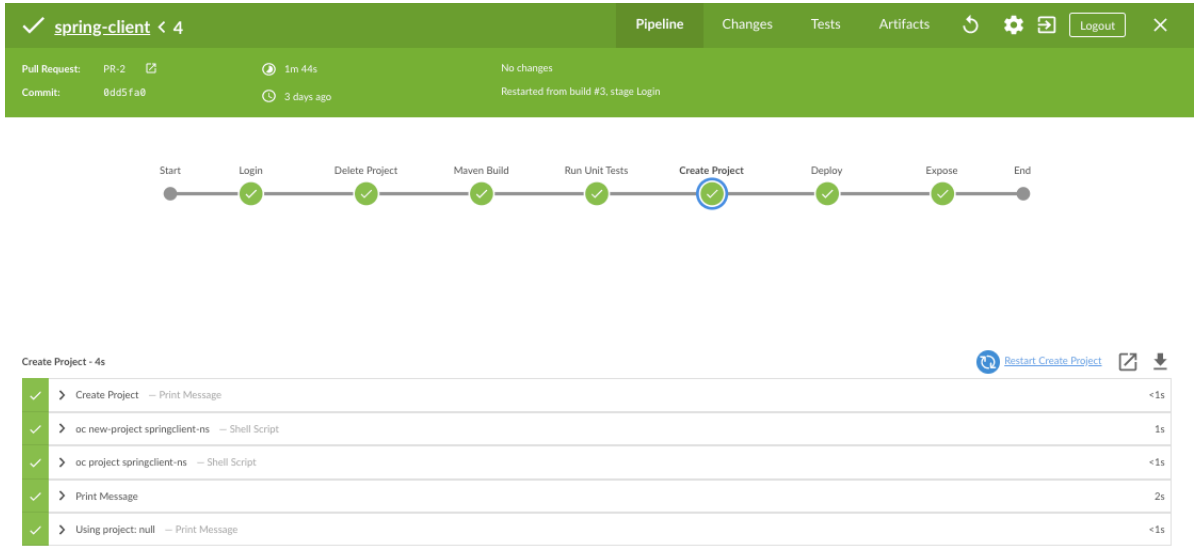
Jenkins

```
    stage("Set KUBE.Context") {
      withCredentials([usernamePassword(credentialsId: "$JENKINS_IKS
_CREDENTIALS", passwordVariable: 'IKS_PW', usernameVariable: 'IKS_
USER')]) {
          sh """
            mkdir ./tmp
            curl -sL https://ibm.biz/idt-installer | bash
            ibmcloud login --skip-ssl-validation -u ${IKS_USER} -
p ${IKS_PW} -a https://${KUBE_URL} -c my-account
         """
      }
    }

    stage("Deploy to KUBE") {
      sh """
        chmod 777 ${DEPLOYMENT_FILE}
        ./${DEPLOYMENT_FILE}
      """
    }
}
```

Github Webhooks



Jenkins Pipeline

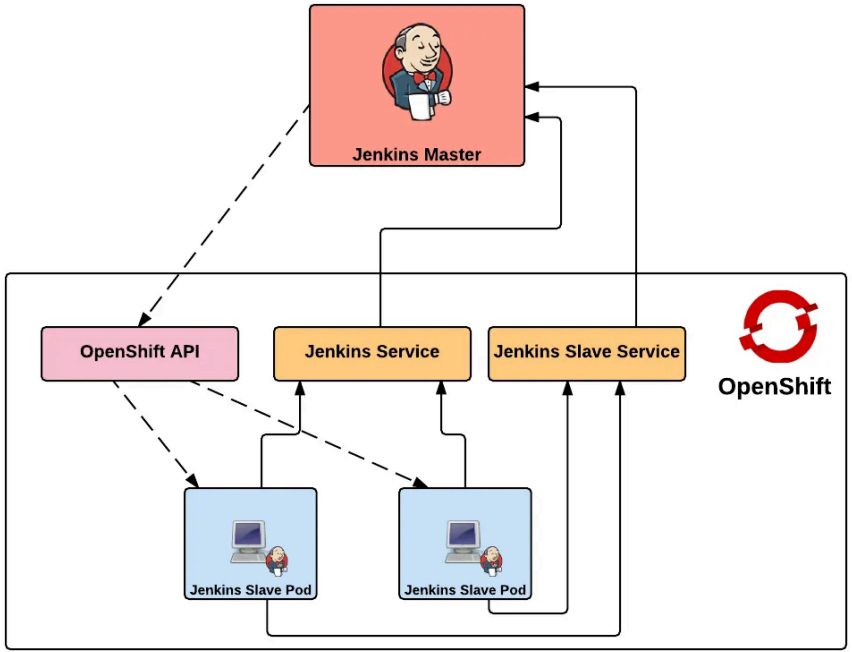


Jenkins external to OpenShift

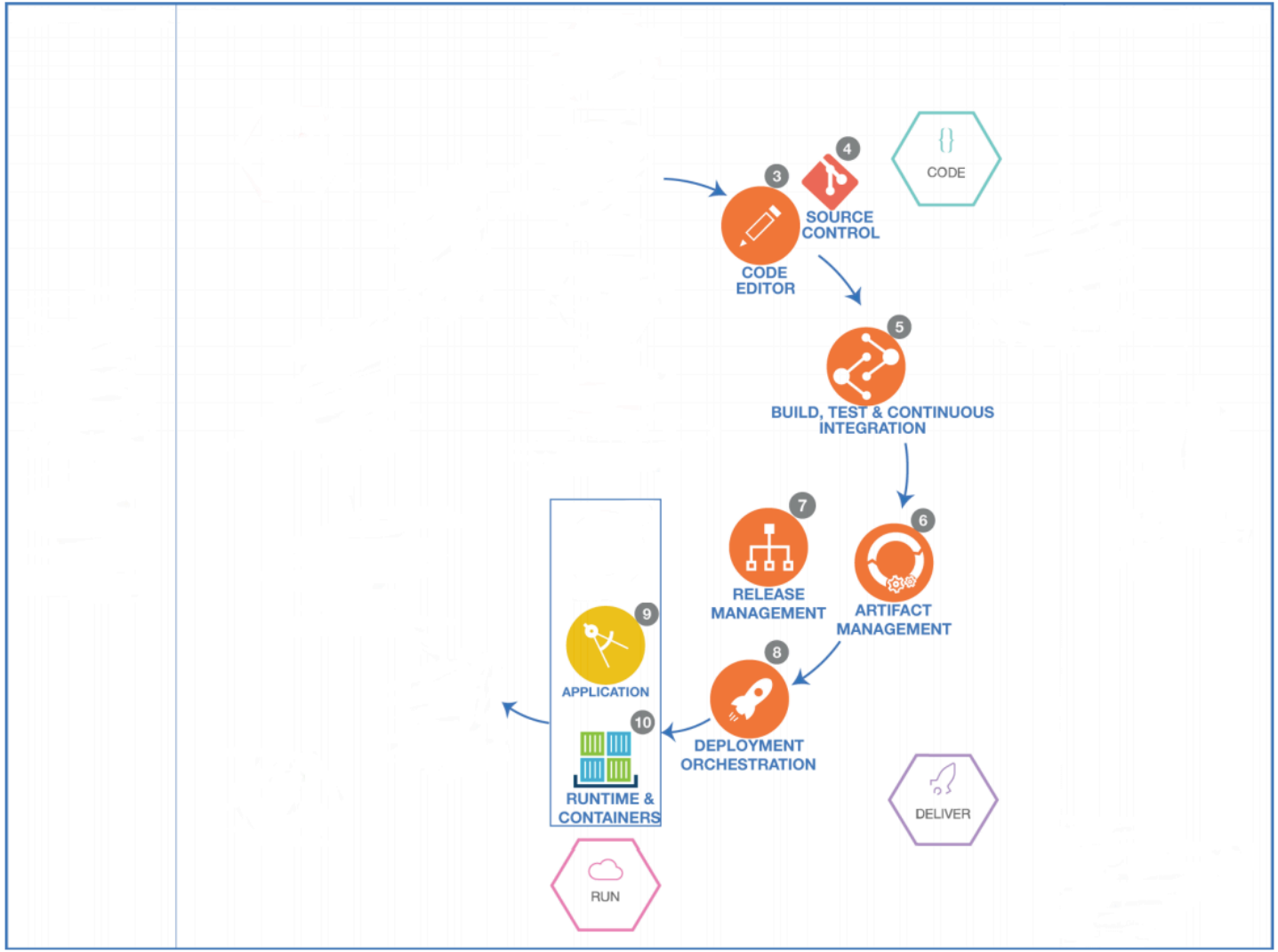
When leveraging the Kubernetes plugin for Jenkins, Jenkins runs dynamic agents in a Kubernetes cluster to dynamically provision slave instances. There must be communication between the Jenkins master and OpenShift:

- Jenkins master communicates with the OpenShift API to manage the lifecycle of slave instances and take advantage of the elasticity OpenShift provides.
- OpenShift and Jenkins communicate via the port exposed by the web console
- Jenkins master communicates via a TCP port used for the JNLP slaves

https://github.com/jenkinsci/kubernetes-plugin



Source: https://blog.openshift.com/jenkins-slaves-in-openshift-using-an-external-jenkins-environment/


CODE
3
4
SOURCE CONTROL
CODE EDITOR
5
BUILD, TEST & CONTINUOUS INTEGRATION
7
RELEASE MANAGEMENT
6
ARTIFACT MANAGEMENT
9
APPLICATION
8
DEPLOYMENT ORCHESTRATION
10
RUNTIME & CONTAINERS
DELIVER
RUN

We can build, test and run on localhost

We can collaborate

We can run anywhere

We pushed our build artifact

We deployed

We automated

We have a CI/CD for Microservices

Okay I knew that...

Plus,

That’s a lot more workload than I used to have, before Microservices

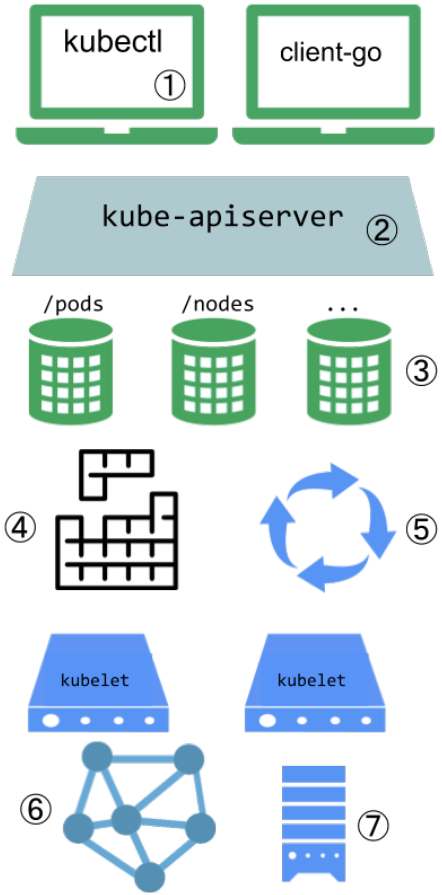
Kubernetes Extensions

Extending Kubernetes

Customization can be divided into:
- configuration,
- Extensions.

You can extend the Kubernetes by using the Controller pattern or the webhook model.
Controllers read a “spec”, do work, and then update the “status”.
When Kubernetes is the client that calls out to a remote service, it is called a Webhook and the remote service is the Webhook Backend.

Extension points:
1. Kubectl, kubectl plugins,
2. Extensions in the apiserver allow authenticating or blocking requests, editing content and handling deletion,
3. Custom Resources (CR) using CustomResourceDefinition API and Operators,
4. Scheduler Extensions,
5. Controllers are often used with CR,
6. Node-level Network Plugins, CNI Plugins or Kubenet plugins,
7. Storage Plugins,



Custom Resource Definition (CRD)

In June 2017, Kubernetes v1.7 introduced Custom Resource Definitions (CRD).

A *Custom Resource (CR)* is an extension of the Kubernetes API that allows you to store an API object of a kind. Even many core Kubernetes functions are now built using custom resources.

A *Custom Resource Definition* (CRD) file defines your own object kinds and lets the API Server handle its lifecycle.

Example: CRD

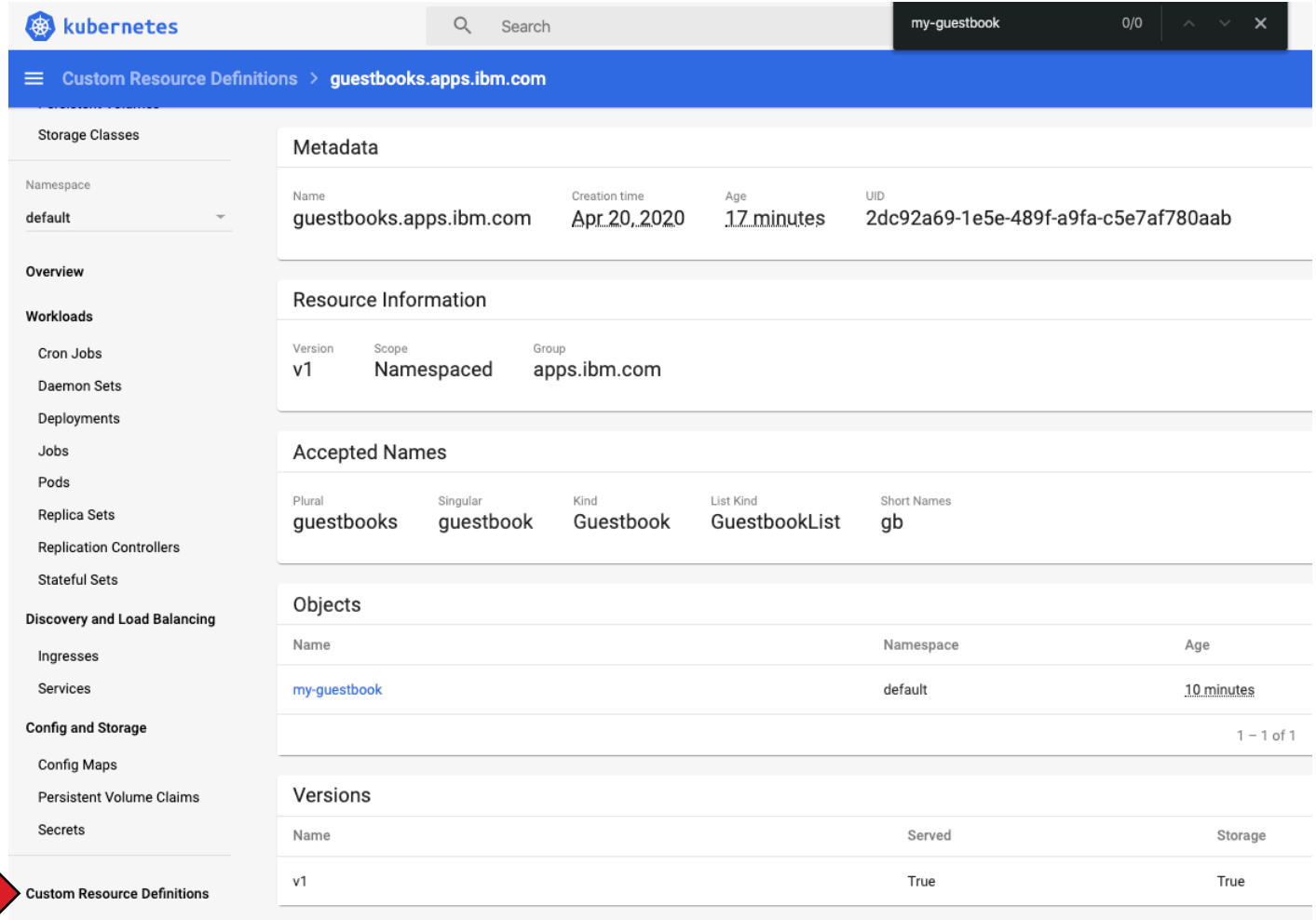
guestbook.yaml

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: guestbooks.apps.ibm.com
spec:
  group: apps.ibm.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                guestbookTitle:
                  type: string
                guestbookSubtitle:
                  type: string
  scope: Namespaced
  names:
    plural: guestbooks
    singular: guestbook
    kind: Guestbook
    shortNames:
    - gb
```

my-guestbook.yaml

```
apiVersion: "apps.ibm.com/v1"
kind: Guestbook
metadata:
  name: my-guestbook
spec:
  guestbookTitle: "The Chemical Wedding of Remko"
  guestbookSubtitle: "First Day"
```

```
% kubectl create -f guestbook-crd.yaml
customresourcedefinition.apiextensions.k8s.io/guestbooks.apps.ibm.com created
% kubectl create -f my-guestbook.yaml
guestbook.apps.ibm.com/my-guestbook created
% kubectl get guestbooks
NAME          AGE
my-guestbook   3m51s
% kubectl describe guestbook my-guestbook
Name:        my-guestbook
Namespace:    default
Labels:      <none>
Annotations:  <none>
API Version:  apps.ibm.com/v1
Kind:        Guestbook
Metadata:
  Creation Timestamp:  2020-04-20T05:40:46Z
  Generation:         1
  Resource Version:    295507
  Self Link:          /apis/apps.ibm.com/v1/namespaces/default/guestbooks/my-guestbook
  UID:                e683caef-33f5-4f83-b54e-85ee163e0abf
Spec:
  Guestbook Subtitle:  First Day
  Guestbook Title:     The Chemical Wedding of Remko
Events:              <none>
```



Operator Pattern

Custom Resources alone let you store and retrieve structured data. The Operator pattern combines custom resources and custom controllers. Combined with a *custom controller*, custom resources provide a true *Declarative API*. You can use custom controllers to encode domain knowledge for specific applications into an extension of the Kubernetes API.

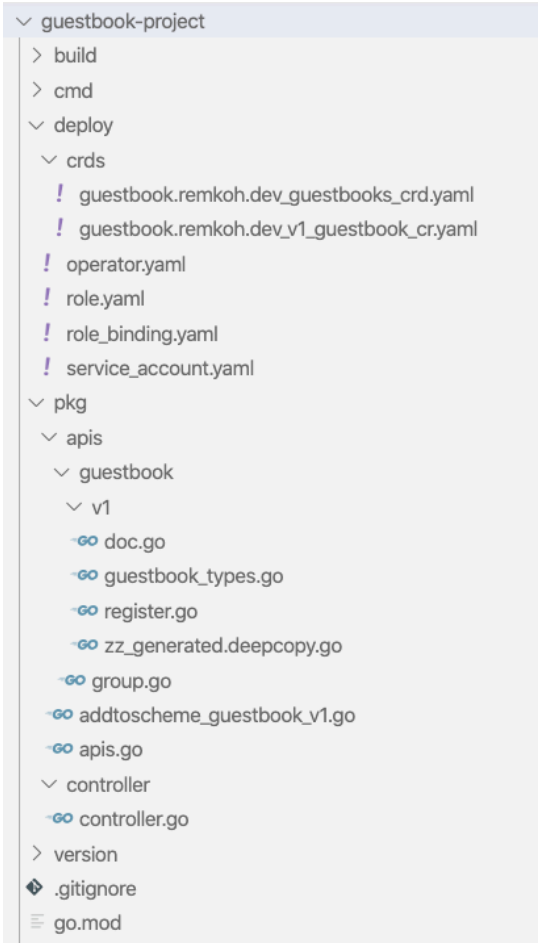
Consider API aggregation if:	Prefer a stand-alone API if:
Your API is Declarative.	Your API does not fit the Declarative model.
Read and write your new types using kubectl.	kubectl support is not required
View your new types in a Kubernetes UI, such as dashboard, alongside built-in types.	Kubernetes UI support is not required.
You are developing a new API.	You already have a program that serves your API and works well.
Accept the format restriction that Kubernetes puts on REST resource paths, such as API Groups and Namespaces. (See the API Overview.)	You need to have specific REST paths to be compatible with an already defined REST API.
Your resources are naturally scoped to a cluster or namespaces of a cluster.	Cluster or namespace scoped resources are a poor fit; you need control over the specifics of resource paths.
You want to reuse Kubernetes API support features.	You don’t need those features.

Operator Framework SDK

```
operator-sdk new $OPERATOR_PROJECT --type go --repo github.com/$DOCKER_USERNAME/$OPERATOR_NAME
operator-sdk add api --api-version=$OPERATOR_GROUP/$OPERATOR_VERSION --kind=$CRD_KIND
operator-sdk add controller --api-version=$OPERATOR_GROUP/$OPERATOR_VERSION --kind=$CRD_KIND
operator-sdk build docker.io/$DOCKER_USERNAME/$OPERATOR_NAME
docker login docker.io -u $DOCKER_USERNAME
docker push docker.io/$DOCKER_USERNAME/$OPERATOR_NAME
sed -i "s|REPLACE_IMAGE|docker.io/$DOCKER_USERNAME/$OPERATOR_NAME|g" deploy/operator.yaml
oc create sa $OPERATOR_PROJECT
oc create -f deploy/role.yaml
oc create -f deploy/role_binding.yaml
oc create -f deploy/crds/${OPERATOR_GROUP}_${CRD_KIND,,}s_crd.yaml
oc create -f deploy/operator.yaml
oc create -f deploy/crds/${OPERATOR_GROUP}_${OPERATOR_VERSION}_${CRD_KIND,,}_cr.yaml
oc get deployment $OPERATOR_PROJECT
oc get pod -l app=example-${CRD_KIND,,}
oc describe ${CRD_KIND,,}s.${OPERATOR_GROUP} example-${CRD_KIND,,}
```

Operator Framework SDK

```
// Reconcile reads state of the cluster for a Guestbook object and makes changes based on the state read and what is in the Guestbook.Spec
// TODO(user): User must modify this Reconcile function to implement their own Controller logic.
This example creates a Pod as an example
func (r *ReconcileGuestbook) Reconcile(request reconcile.Request) (reconcile.Result, error) {
   ...
   // Fetch the Guestbook instance
   instance := &guestbookv1.Guestbook{}
   ...
   // Define a new Pod object
   pod := newPodForCR(instance)
   ...
}
```



Application (CRD)

https://github.com/kubernetes-sigs/application

```
apiVersion: app.k8s.io/v1beta1
kind: Application
metadata:
  name: "guestbook"
  labels:
    app.kubernetes.io/name: "guestbook"
spec:
selector:
matchLabels:
app.kubernetes.io/name: "guestbook"
componentKinds:
- group: v1
kind: Deployment
- group: v1
kind: Service
descriptor:
type: "guestbook"
keywords:
- "gb"
- "guestbook"
links:
- description: Github
url: "https://github.com/IBM/guestbook"
version: "0.1.0"
description: "The Guestbook application is an example app to demonstrate key
Kubernetes functionality."
maintainers:
- name: IBM Developer
email: developer@ibm.com
owners:
- name: IBM Developer
email: developer@ibm.com
```

Build from Source

Source-to-Image (S2I)

Source-to-Image (S2I) builds reproducible container images from source code:

- From a builder image,
- Inject source code into a known directory,
- Create a runnable setup from source code,
- Commit the new container and set the image entrypoint.

For compiled languages, S2I creates a builder image and a runtime image.

Deploying a Spring Boot Application to Kubernetes

```
$ docker pull registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift
$ s2i build . registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift s2i-spring-client
$ oc project s2i-spring-client-ns
$ oc new-app --name springclient registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift~.
  --strategy=source --allow-missing-images --build-env='JAVA_APP_JAR=hello.jar'
$ oc expose svc/springclient
$ curl -X GET  'http://<openshift-url>/api/hello?name=pluto'
```

https://code.remkohde.com/openshift/builds/source-to-image-s2i/build-and-run-from-source

Add Webhook

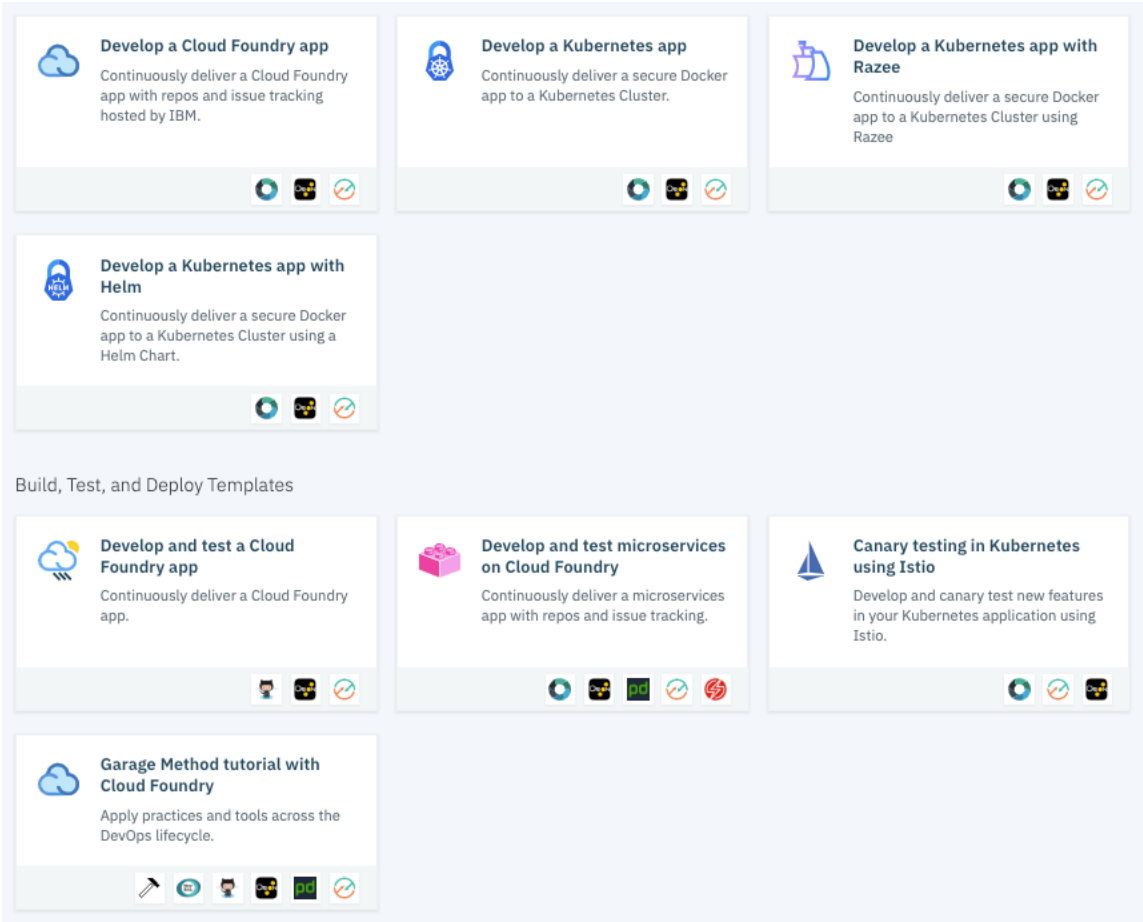
Get Github webhook

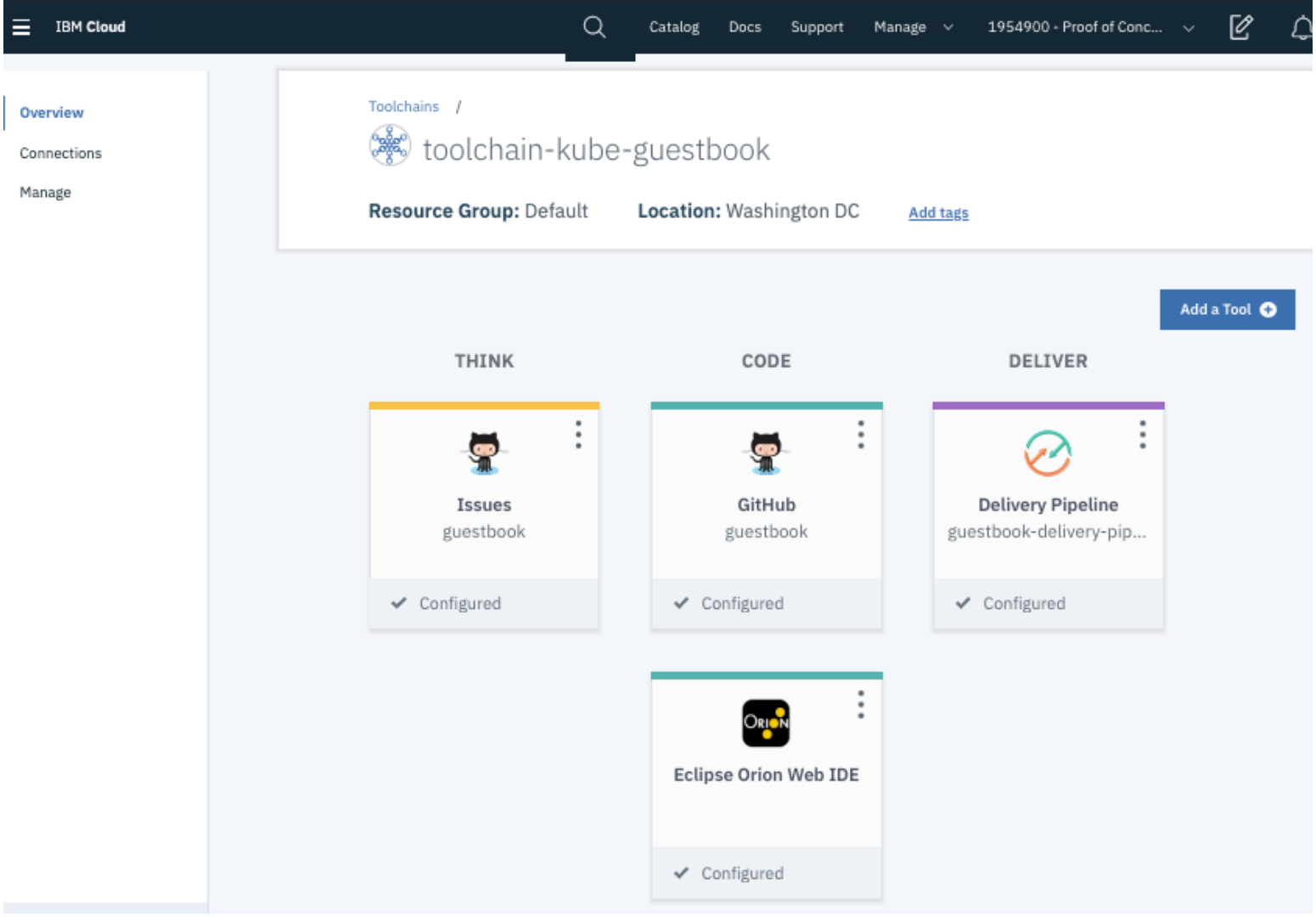
```
$ oc status
In project s2i-spring-client-ns on server https://c100-e.us-south.containers.cloud.ibm.com:30403
http://springclient-s2i-spring-client-ns.cda-openshift-cluster-9c1e3bfb0c12345cf678a9ca7dd1e060-0001.us-south.
containers.appdomain.cloud to pod port 8080-tcp (svc/springclient)
  dc/springclient deploys istag/springclient:latest <-
    bc/springclient source builds https://github.com/remkohdev/spring-client.git#master on istag/openjdk18-openshift:latest
    deployment #1 deployed 24 hours ago - 1 pod

$ oc describe bc/springclient |  grep -i webhook
...
Webhook GitHub:
    URL: https://c100-e.us-south.containers.cloud.ibm.com:30403/apis/build.openshift.io/v1/namespaces/s2i-spring-client-ns/
buildconfigs/springclient/webhooks/<secret>/github
Webhook Generic:
```

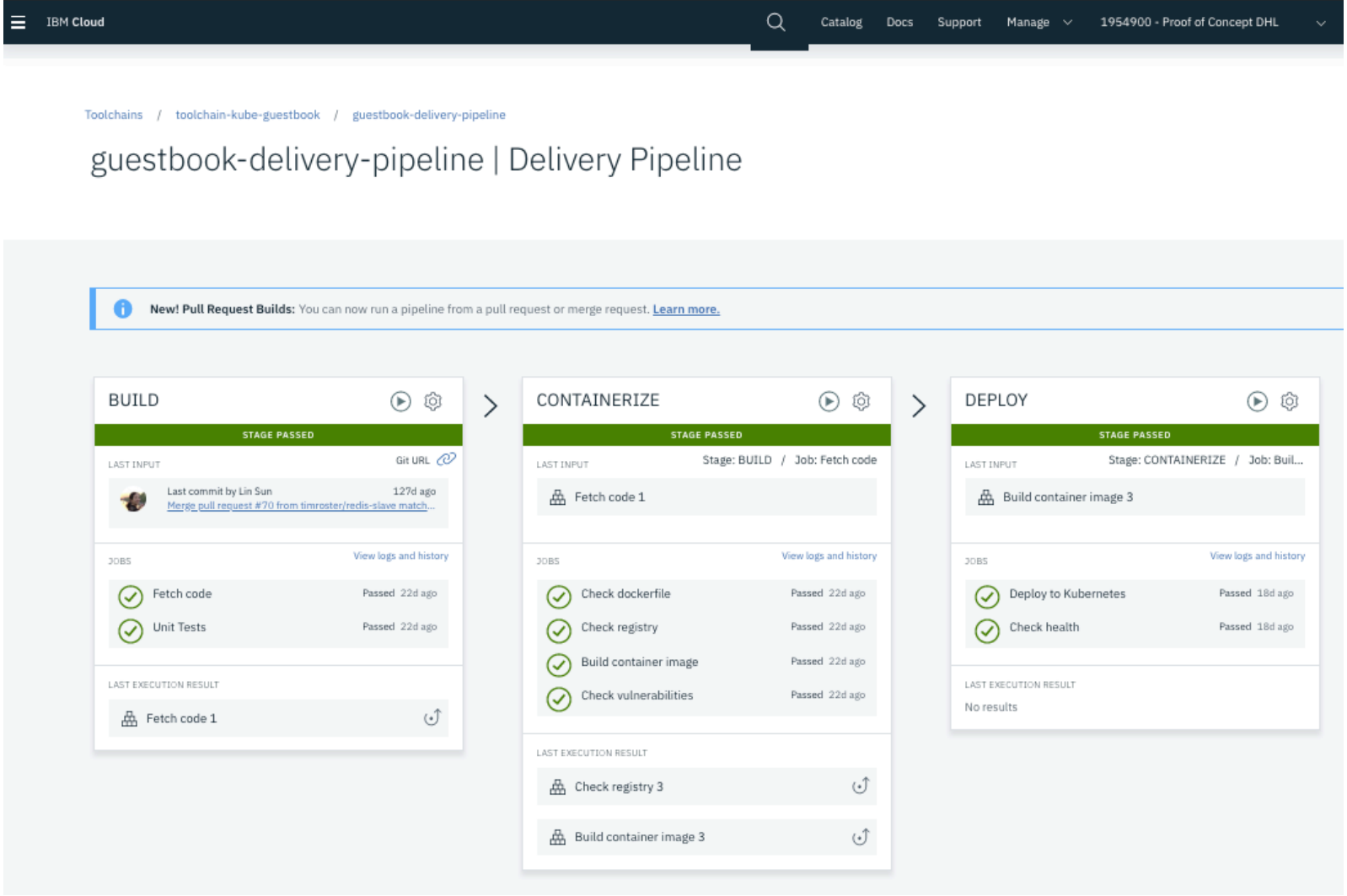
You can find the URL secret for the Github webhook as a part of the full Route path,
e.g. `9c1e3bfb0c12345cf678a9ca7dd1e060-0001`

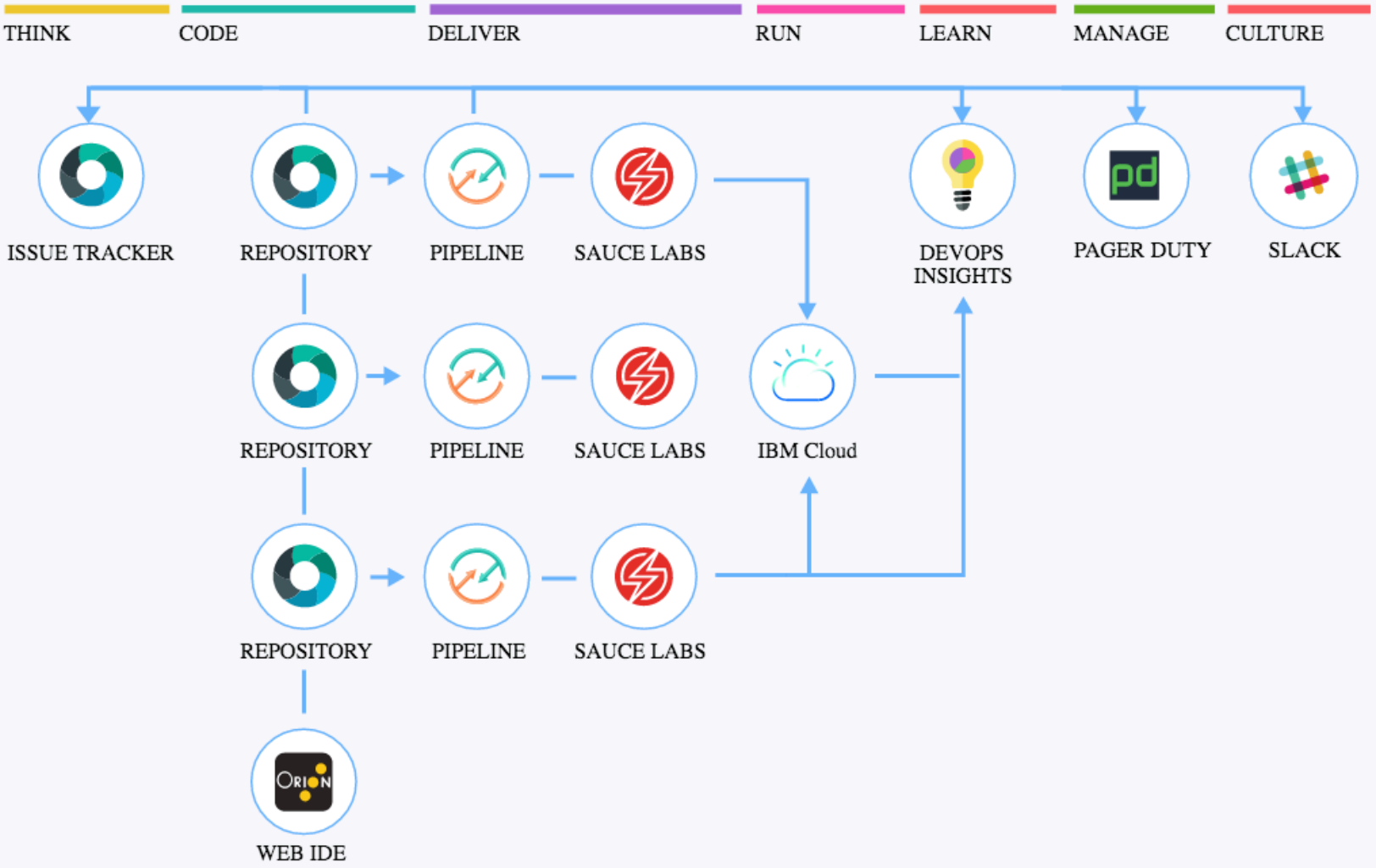
Open Toolchain (OTC) Templates

https://github.com/open-toolchain/sdk/wiki



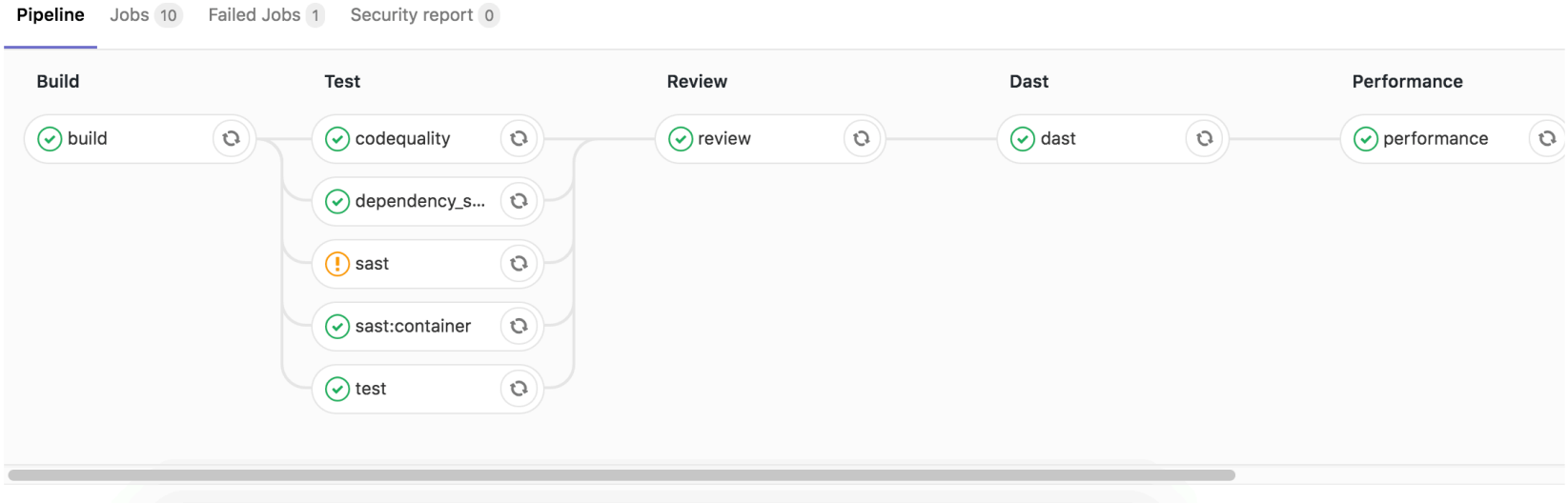

IBM Cloud
Catalog
Docs
Support
Manage
1954900 - Proof of Conc...
Overview
Connections
Manage
Toolchains /
toolchain-kube-guestbook
Resource Group: Default
Location: Washington DC
Add tags
Add a Tool
THINK
CODE
DELIVER
Issues
guestbook
Configured
GitHub
guestbook
Configured
Delivery Pipeline
guestbook-delivery-pip...
Configured
Eclipse Orion Web IDE
Configured


IBM Cloud
Catalog
Docs
Support
Manage
1954900 - Proof of Concept DHL
Toolchains / toolchain-kube-guestbook / guestbook-delivery-pipeline
guestbook-delivery-pipeline | Delivery Pipeline
New! Pull Request Builds: You can now run a pipeline from a pull request or merge request. Learn more.
BUILD
STAGE PASSED
LAST INPUT
Git URL
Last commit by Lin Sun
127d ago
Merge pull request #70 from timroster/redis-slave match...
JOBS
View logs and history
Fetch code
Passed 22d ago
Unit Tests
Passed 22d ago
LAST EXECUTION RESULT
Fetch code 1
CONTAINERIZE
STAGE PASSED
LAST INPUT
Stage: BUILD / Job: Fetch code
Fetch code 1
JOBS
View logs and history
Check dockerfile
Passed 22d ago
Check registry
Passed 22d ago
Build container image
Passed 22d ago
Check vulnerabilities
Passed 22d ago
LAST EXECUTION RESULT
Check registry 3
Build container image 3
DEPLOY
STAGE PASSED
LAST INPUT
Stage: CONTAINERIZE / Job: Buil...
Build container image 3
JOBS
View logs and history
Deploy to Kubernetes
Passed 18d ago
Check health
Passed 18d ago
LAST EXECUTION RESULT
No results


THINK
CODE
DELIVER
RUN
LEARN
MANAGE
CULTURE
ISSUE TRACKER
REPOSITORY
PIPELINE
SAUCE LABS
DEVOPS INSIGHTS
PAGER DUTY
SLACK
REPOSITORY
PIPELINE
SAUCE LABS
IBM Cloud
REPOSITORY
PIPELINE
SAUCE LABS
WEB IDE

Gitlab Auto DevOps

- Uses Heroku buildpacks,
- Build-in pipeline and configuration for Git and Kubernetes,
- Includes code quality, vulnerability, licenses, security checks, buildpack tests



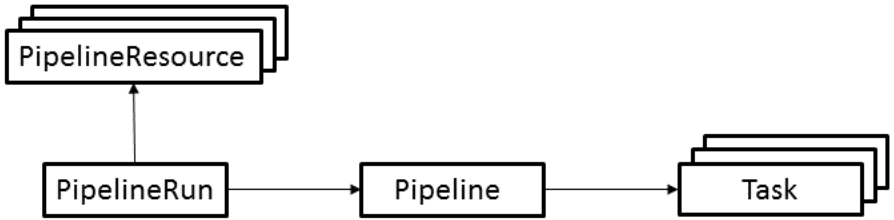
Tekton

Tekton

Tekton is open-source project, born out of Knative Build, and part of the CD Foundation, a Linux Foundation project. The Tekton Pipelines project has support and active commitment from companies, including IBM, Red Hat, Google, and CloudBees.

Tekton

- A **PipelineResource** defines an object that is an input (such as a git repository) or an output (such as a docker image) of the pipeline.
- A **PipelineRun** defines an execution of a pipeline. It references the **Pipeline** to run and the **PipelineResources** to use as inputs and outputs.
- A **Pipeline** defines the set of **Tasks** that compose a pipeline.
- A **Task** defines a set of build steps such as compiling code, running tests, and building and deploying images.



Tekton

```
apiVersion: tekton.dev/v1alpha1
kind: EventListener
metadata:
  name: hello-listener
spec:
  triggers:
    - binding:
       name: hello-trigger-binding
    template:
      name: hello-trigger-template
```

```
apiVersion: tekton.dev/v1alpha1
kind: TriggerTemplate
metadata:
  name: hello-trigger-template
spec:
  resources:
    - name: source-repo
      resourceRef:
        name: source-repo
        type: git
  resourcetemplates:
    - apiVersion: tekton.dev/v1alpha1
      kind: PipelineRun
      metadata:
        name: pipelinerun-$(uid)
      spec:
        pipelineRef:
          name: hello-pipeline
        resources:
          - name: source-repo
            resourceRef:
              name: source-repo
```

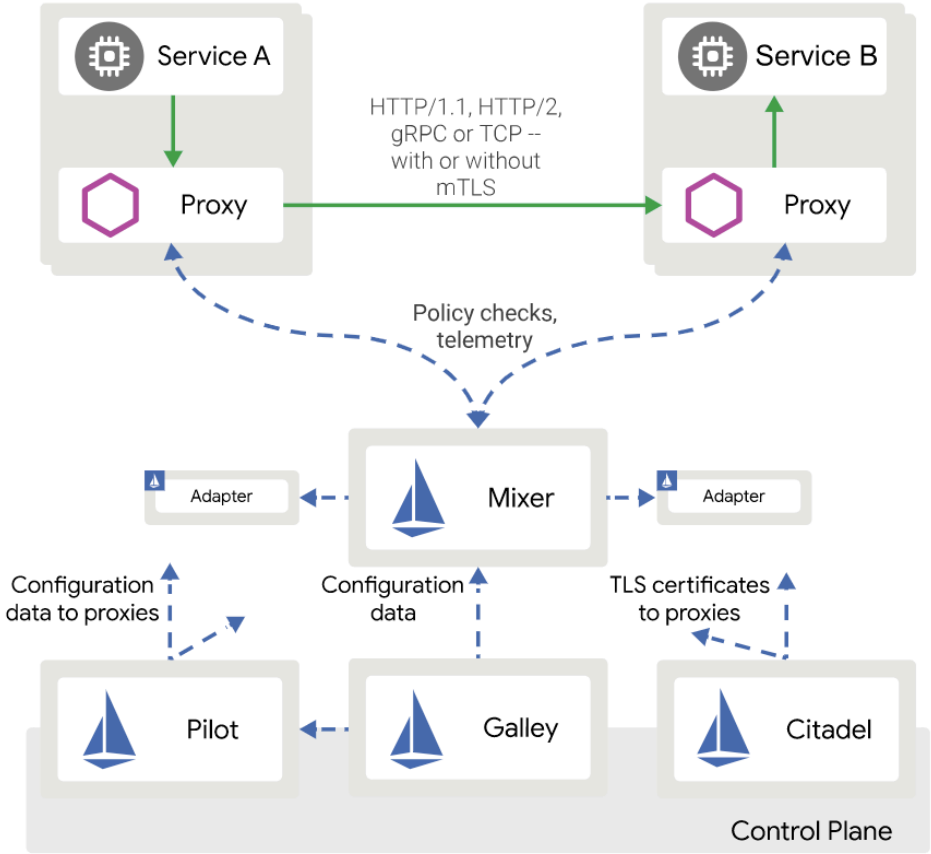
Istio

Istio simplifies configuration of :

- circuit breakers,
- timeouts,
- retries,
- A/B testing with Traffic Shifts,
- canary rollouts,
- staged rollouts with percentage-based traffic splits,
- failure recovery features.

Istio's traffic management model relies on the Envoy proxies that are deployed with your services. All traffic (data plane traffic) is proxied through Envoy, making it easy to direct and control traffic around your mesh.

The Istio API is specified using Kubernetes custom resource definitions (CRDs),



Traffic Shifts with VirtualService

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v3
```

```
- match:
  - headers:
      user-agent:
        regex: '^.*(Android|iPhone).*$'
```

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
      weight: 50
    - destination:
        host: reviews
        subset: v3
      weight: 50
```

Lab Time