Jango Fett

Spawning developer environments

A not so long time ago
In a galaxy nearby …

A rebel start-up wanted to build something incredible

But they were limited by the number of dev environments

So they reached out to someone who could save them, their last hope … Me !

About Me: Yash Mehrotra

Backend, Infrastructure & Platform

Software Engineer @ Flanksource

Prev. Lummo, MindTickle, BlinkIt

Small detour: Kubernetes 101

- A container-orchestration system for automating application deployment, scaling, and management.
- Works on declarative configuration principle
- You apply YAML or JSON manifests and the system ensures the desired state is achieved

Kubernetes 101: Terminology

Namespace: A mechanism to provide virtual isolation for resources

Pods: Group of one or more containers

Deployment: Maintain a set of replica pods

ConfigMap: Store key-value pairs which can be used by pods

Kubernetes 101: Sample Deployment Manifest

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  template:
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Kubernetes 101: Sample Deployment Manifest

```
$ kubectl apply -f manifest.yml
Deployment "nginx" created

$ kubectl get pods --namespace default
NAME                    READY     STATUS    RESTARTS
nginx-7d5b423-gd54   1/1       Running   0
nginx-7d5b423-rt41   1/1       Running   0
nginx-7d5b423-lb4y   1/1       Running   0
```

Kubernetes 201: CRDs

- Custom Resource Definition (CRD) allows you to define your own custom resource for the Kubernetes API
- Write your own controller to manage the system behaviour when interacting with your CRD
- Easy way to extend Kubernetes API

Kubernetes 201: CRDs

```yaml
apiVersion: stable.example.com/v1
kind: Ping
metadata:
  name: example-com
spec:
  destination: example.com
  ipv6: true
```

Kubernetes 201: CRDs

```
func Reconcile(ctx, object) {
  dst := object.Spec.Destination
  isIpV6 := object.Spec.IpV6

  p := ping.NewPinger(dst, isIpV6)
  p.Count := 5
  p.Run()
}
```

The problem

- Current setup:
  - 1 production environment
  - 1 staging environment
  - 2 dev environments
- Wait to deploy your changes if someone else is using the dev environment
- Dev environments kept changing and lost parity with prod and staging

The problem

- No process for experimenting with things that may break dev environments
- Platform team started becoming the bottleneck for changes
- QA team had to wait for all the features to be merged in a single release to even begin testing

Episode I

The search begins

Goals

- Intuitive to use
- Kubernetes first approach
- Seamlessly integrate with our current system
- Supports parallel development on multiple services
- Isolation with other environments

No perfect fit

- Every tool solved a specific problem but not all
- Most of them had a learning curve
- Many were in active development and not stable

Will we embrace any ?

- Tons of glue code required to make any tool work
- They came with their own domain knowledge
- No way to fix missing features

What if …
We build our own tool ?

I will do what I must

Breaking down the problem

- Services are easy to deploy since they are stateless
- For absolute isolation, state needs to be isolated
- Make it easier to deploy stateful dependencies like
  - PostgreSQL
  - Redis
  - Object storage service
  - Pub/Sub Queues

Guiding principles

- Use CRDs to represent everything
- Leverage kubernetes' lifecycle management
- Make it generic to support a range of projects
- Iteratively add features incorporating user feedback

Episode II

Building the machine

Going with Go

- Robust and easy to ship
- Amazing ecosystem for kubernetes tooling
- Statically typed
- and a great community

Lets build some cool sh*t

Kubebuilder

- Framework for building Kubernetes APIs using CRDs
- Uses code-generation
- Natively works with Kubernetes source libraries
- Uses a Reconcile loop to reciprocate Kubernetes' declarative philosophy

Environment CRD

Creates a namespace and gives the users admin access to it

```yaml
apiVersion: jango-fett.bukukas.io/v1alpha1
kind: Environment
metadata:
  name: sample
spec:
  users:
    - yash@lummo.com
    - pair-programmer@lummo.com
  config:
    foo: bar
    baz: foo
```

Environment CRD

```
func Reconcile(ctx, object) {
  if !k8s.DoesObjectExist(object.Metadata.Name) {
    // Cleanup process
    k8s.DeleteNamespace(...)
    return
  }

  k8s.CreateNamespace(...)
  k8s.CreateConfig(...)
  k8s.ConfigureAuth(...)
}
```

PostgreSQL CRD

Creates a PostgreSQL database in an existing PostgreSQL Server

```yaml
apiVersion: jango-fett.bukukas.io/v1alpha1
kind: PostgreSQL
metadata:
  name: notifications
  namespace: jf-sample
spec:
  extensions:
    - uuid
```

PostgreSQL CRD

```
func Reconcile(ctx, object) {
  if !k8s.DoesObjectExist(object.Metadata.Name) {
    // Cleanup process
    database.Exec("DELETE DATABASE ?", getDBName(object))
    return
  }

  database.Exec("CREATE DATABASE ?", getDBName(object))
  database.Exec("CREATE USER ?", getDBUser(object))

  // Applications can read these secrets to connect
  k8s.CreateSecretsForDatabase(object)
}
```

Redis CRD

Creates a Redis container in the namespace

```yaml
apiVersion: jango-fett.bukukas.io/v1alpha1
kind: Redis
metadata:
  name: common
  namespace: jf-sample
spec: {}
```

Redis CRD

```
func Reconcile(ctx, object) {
  redisK8sObject := generateRedisK8sObject(object)
  if !k8s.DoesObjectExist(object.Metadata.Name) {
    // Cleanup process
    k8s.DeleteObject(redisK8sObject)
    return
  }

  k8s.CreateObject(redisK8sObject)
  k8s.CreateSecretsForRedis(redisK8sObject)
}
```

Kubebuilder: Testing

- Great testing experience
- Launches the control plane to simulate kubernetes
- Uses Ginkgo framework
- You write Behaviour Driven Tests

Kubebuilder: Testing

```
Context("PostgreSQL object lifecycle", func() {

  It("Should accept the PostgreSQL CRD", func() {
    By("By creating a new PostgreSQL Object")
    Expect(k8sClient.Create(ctx, postgresObj)).Should(Succeed())
  })
}
```

Kubebuilder: Testing

```
Context("PostgreSQL object lifecycle", func() {

  It("Should create all the resources", func() {
    By("Creating Secret object")
    // Check if object is created

    By("Creating database with user")
    // Query metadata database

    By("Checking status of PostgreSQL object")
    // Get the desired state by continuous querying
    Eventually(func() bool {
      // Logic
    }, timeout, interval).Should(BeTrue())
  }
}
```

Deploying a complete application

- Let's deploy the logistics service
- Assume we have to fix a bug
- Then need to test the entire flow on the mobile app

Deploying a complete application

```yaml
kind: Environment
metadata:
  name: logistics-fix-silly-bug
spec:
  users:
    - yash@lummo.com
---
kind: PostgreSQL
metadata:
  name: logistics-api
---
kind: Redis
metadata:
  name: logistics-api
```

```yaml
kind: Deployment
metadata:
  name: logistics-api
---
kind: Deployment
metadata:
  name: logistics-worker
---
kind: Service
metadata:
  name: logistics-api
---
kind: ConfigMap
metadata:
  name: logistics-api
```

Deploying a complete application: Behind the scenes

- Environment CRD creates a new namespace
- PostgreSQL database is created in the dev-postgresql server
- Redis StatefulSet is created
- Logistic API and Worker applications are deployed

Deploying a complete application: Your own paradise

Deploying a complete application: Workflow

- Deploy your code in a prod-like environment
- You get an endpoint like ***logistics-fix-bug.dev.example.com***
- Use this URL in the debug build of the mobile app
- Push to production once the changes are working

But, there was a problem …
Not everyone was using it

Episode III

The Adoption Menace

How was the Developer Experience ?

- Devs used a shell script to apply the manifests
- Script was a bit flaky
- No feedback for when something went wrong
- Difficult to update since user's ran it from their local

Harsh truth about software

Whatever you build in this life, it's not legendary

unless your customers use it

Back to the drawing board

- Did user research to find out the real pain points
- Figured out what they expected from the tool
- Surveyed them to get all the common use cases

Taking a step back

We built an amazing engine, but people do not buy engines

They buy cars

Hello CLI

- Create a CLI tool for the devs
- Descriptive help section for commands and subcommands
- Encapsulate all the common use cases

Going with Go

- Viper is an excellent library for CLI applications
- Single static binary, compile anywhere, run everywhere
- Easy to distribute and update

```
go install github.com/…/…
```

Dawn of a new era

```
$ jf create env logistics-fix-bug
Created new environment "logistics-fix-bug"

$ jf deploy logistics-fix-bug logistics-api
Deployed "logistics-api"
Deployed "logistics-worker"
Created service "logistics-api"
Deployed PostgreSQL "logistics-api"
Deployed Redis "logistics-api"
```

Dawn of a new era

```
$ jf delete env logistics-fix-bug
Deleted environment "logistics-fix-bug"

$ jf restart logistics-fix-bug logistics-worker
Restarted "logistics-worker"

$ jf config update logistics-fix-bug key1=value1
Updated config
```

Devs loved the new changes

Sooraj

I tried jango fett and it was brilliant, the tooling is super slick

Abhishek

Bro, this thing is epic, made my work so easy now

Sumeeti

Thanks for the cli, you literally saved hours of my time

More than 80% of the features shipped that quarter were built using Jango Fett

… and the balance was restored

Takeaways

- Its okay to build in-house tools
- Go is great for extending upon Kubernetes
- Gather early feedback and iterate
- Customer obsession is key

The End

yashmehrotra.com

@yashm95