Kapitan

Keep your ship together

About

Ricardo Amaro (@ramaro)
Software Engineer at DeepMind Health
DevOps Team

We’re hiring! https://deepmind.com/careers/

There is no Artificial Intelligence in this presentation

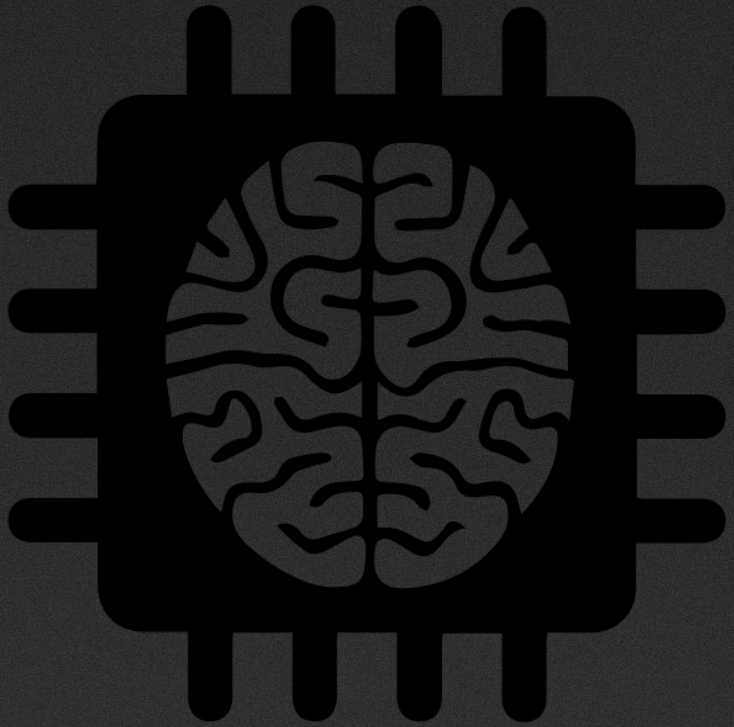


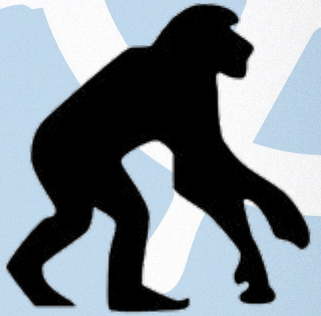
Image Credit - Krisztián Mátyás

kubectl run

```
$ kubectl run nginx-project1 --image=nginx:1.9.1
$ kubectl set image deployment/nginx-project1 nginx=nginx:1.9.2
```

Naive approach

Imperative



kubectl apply

```
$ kubectl apply -f repository/project1/nginx.yml
```

Initial approach to use simple files in a repository

Simple to manage and reproduce on a small project

Harder for large projects

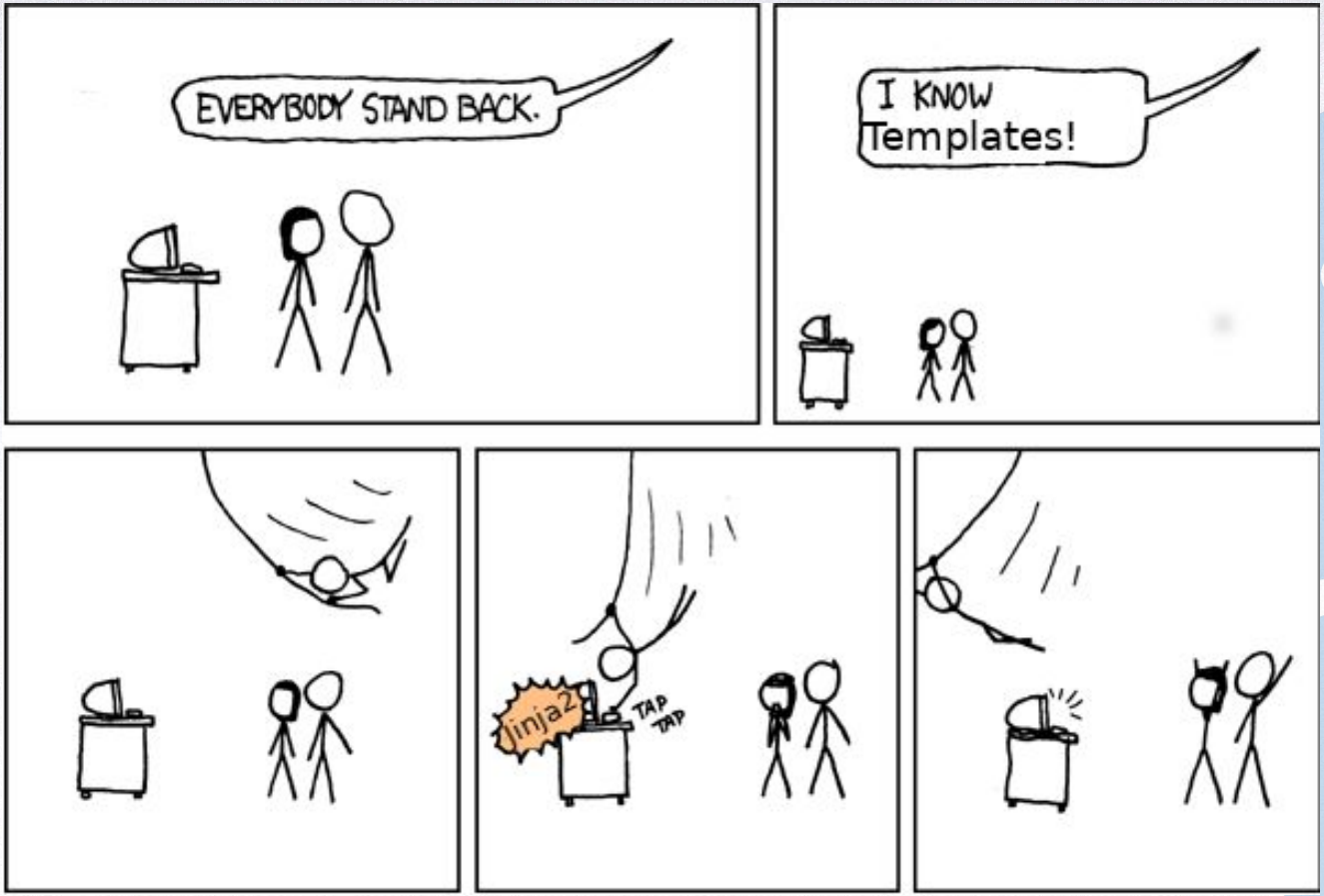


Image Credit - Randall Munroe

Jinja2 Templating

Hey, I know templates!

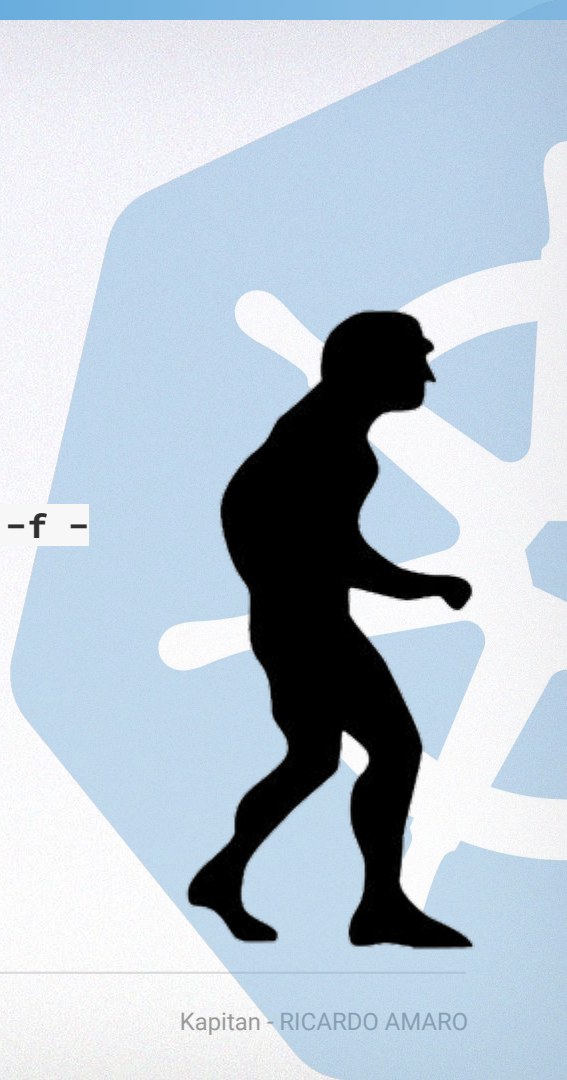
- YAML file containing vars
- Template files with Jinja2 and YAML
- Jinja2 command line tool

```
$ j2 project1.vars.yaml repository/nginx.yml | kubectl apply -f -
```

First attempt to templating using Jinja

Need a sidecar container?

```
{{ sidecarcontainer | indent( width=8) }}
```



Helm

Community support

- Packaged charts
- Basic sharing of variables within chart
- Tiller steering your boat

First signs that we can improve things:

```
{{ sidecarcontainer | indent(width=8) }}
```

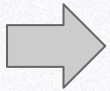
```
{{ sidecarcontainer | indent(width=10) }}
```

Jsonnet

Templating language for structured data - jsonnet.org

- Produces JSON/YAML
- Object oriented
- Has an import system
- JSON/YAML is its single purpose
- Not enough (and that's OK)

```
// Jsonnet Example
{
    person1: {
        name: "Alice",
        welcome: "Hello " + self.name + "!",
    },
    person2: self.person1 { name: "Bob" },
}
```



```
{
   "person1": {
      "name": "Alice",
      "welcome": "Hello Alice!"
   },
   "person2": {
      "name": "Bob",
      "welcome": "Hello Bob!"
   }
}
```

Image Credit - SpaceX

Why Kapitan

- Declarative
- Reusable components
- Remove ambiguity from contexts
- Manage variables/components
- Templating
- Produce up to date rich documentation/tooling
- Promote D.R.Y.
- Encrypted secrets
- Source control focused

Targets

- Where and what to deploy
- Single deployable unit
- Unify the concept of context with a namespace in a cluster
- Only definition of the location of an environment
- Allow importing components
- Allow overriding default variables

Variable Inventory

Reclass - github.com/madduck/reclass

- Hierarchical representation of variables and components

  (similar to Puppet Hiera)
- Simple inheritance
- Interpolation
- Defines targets (leaves in the hierarchical tree)
- Defines classes (sets of variables used on the targets)

components/myapp.yml

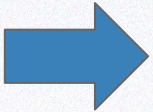
```
myapp:
  name: app1
  image: eu.gcr.io/tuna/img-a:v1
  args:
    app_name: ${myapp:name}
    verbose: true
...
```

components/otherapp.yml

```
otherapp:
  name: otherapp1
  image: eu.gcr.io/tuna/img-b:v1
  verbose: true
  args:
    app_name: ${otherapp:name}
    verbose: ${otherapp:verbose}
...
```

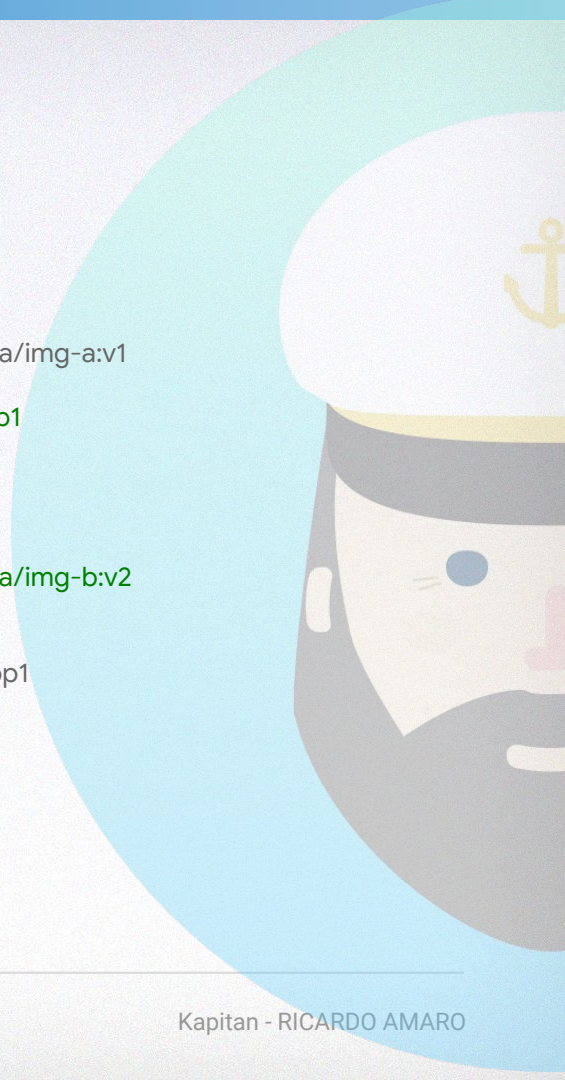
targets/prod-tuna.yml

```
classes:
  - components.myapp
  - components.otherapp

parameters:
  myapp:
    name: prodapp1

  otherapp:
    verbose: false
    image: eu.gcr.io/tuna/img-b:v2
...
```



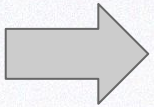
prod-tuna

```
myapp:
  name: prodapp1
  image: eu.gcr.io/tuna/img-a:v1
  args:
    app_name: prodapp1
    verbose: true

otherapp:
  otherapp: otherapp1
  image: eu.gcr.io/tuna/img-b:v2
  verbose: false
  args:
    app_name: otherapp1
    verbose: false
...
```



My App

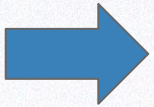
```
local myContainers = [
   { name: "nginx", image: "nginx:1.10" },
];

{
   my-app: {
     apiVersion: "apps/v1beta1",
     kind: "Deployment",
     metadata: { name: "my-app" },
     spec: {
        replicas: 1,
        template: {
           metadata: {
              labels: { app: "my-app" },
           },
           spec: {
              containers: myContainers,
...
```



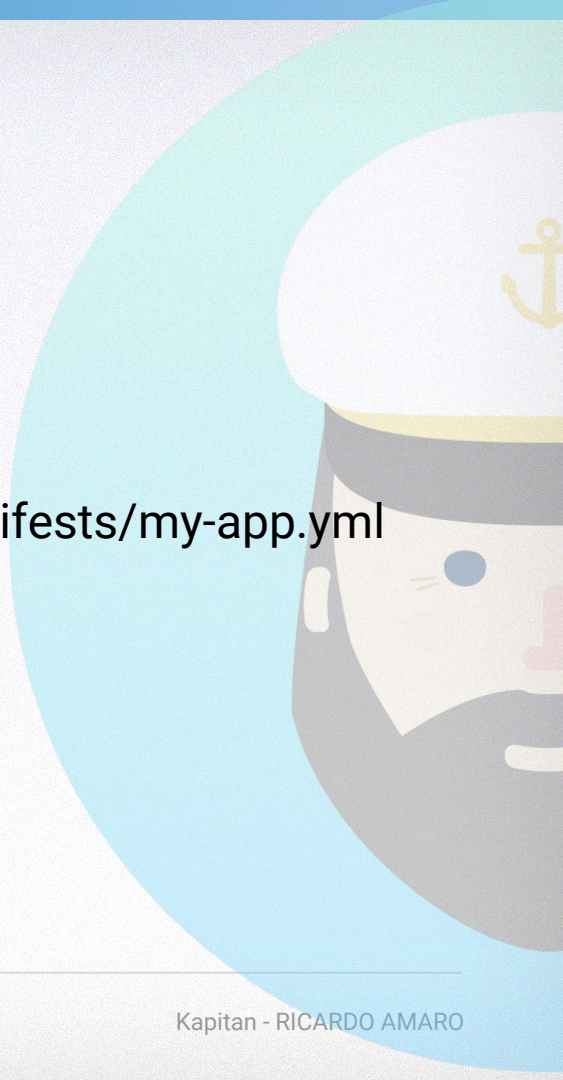
my-app.yml

My App with vars

```
local kap = import "lib/kapitan.libjsonnet";
local inv = kap.inventory();

local myContainers = [
  { name: "nginx", image: inv.parameters.nginx.image },
];
{
 my-app: {
   apiVersion: "apps/v1beta1",
   kind: "Deployment",
   metadata: {
     name: "my-app",
     namespace: inv.parameters.namespace,
    },
   spec: {
      replicas: inv.parameters.myapp.replicas,
      template: {
         metadata: {
           labels: { app: "my-app" },
         },
        spec: {
          containers: myContainers,
....
```

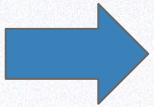


target1/manifests/my-app.yml



My App docs from vars

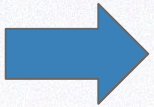
```
{% set i = inventory.parameters %}


# Welcome to the README!

Target *{{ i.target }}* is running:

* {{ i.myapp.replicas }} replicas of *myapp* running
  nginx image {{ i.nginx.image }}
* on cluster {{ i.cluster.cluster }}
```



target1/README.md

My App scripts with vars

```
#!/bin/bash

{% set i = inventory.parameters %}
{% set cluster = i.cluster %}


kubectl config set-context {{i.target}} --cluster {{cluster.name}}
\ --user {{cluster.user}} --namespace {{i.namespace}}

kubectl --context {{i.target}} apply -f manifests/
```



target1/scripts/apply.sh

Kapitan

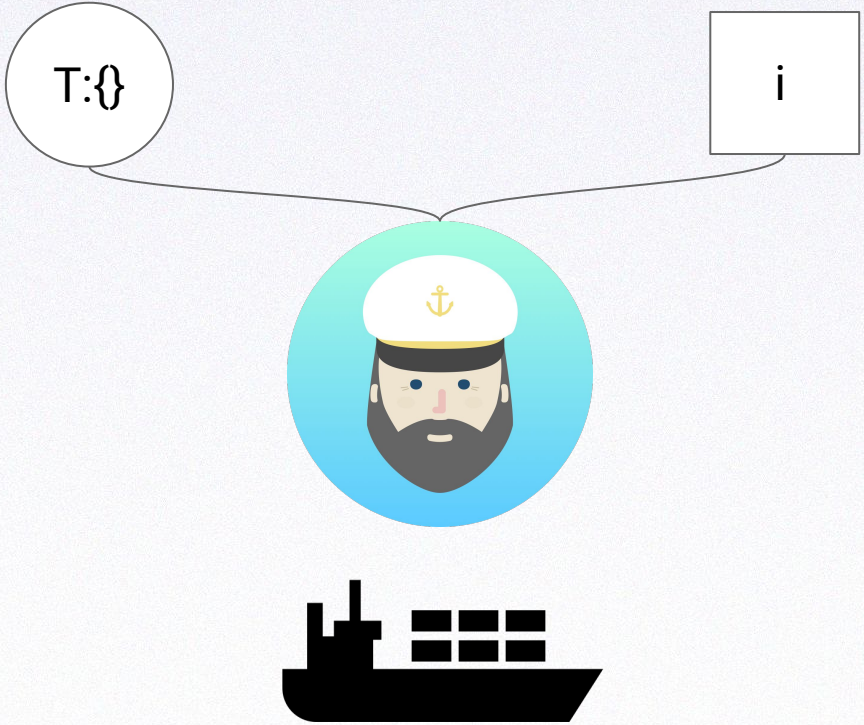


Image Credit - Sascha Elmers

Demo

https://github.com/ramaro/kapitan-examples

Q&A

https://github.com/deepmind/kapitan

#kapitan on kubernetes.slack.com

We’re hiring! https://deepmind.com/careers/

Thanks!