Posted by 3 years ago

What are some best practices for organizing kubernetes yaml when dealing with multiple microservices in a project?

I am relatively new to using kubernetes and microservices. I've used docker for some time and the services themselves I am comfortable building inside containers.

Now that I have several "microservices" as python apps, expressjs apps, connected either via REST calls or through mechanisms like PubSub, how do folks organize the components into a project? I'm looking for approaches that allow good change management practices, documentation, integrated logging, and CI/CD/testing.

I realize this is asking for broad advice, but I get the sense that there are multiple approaches and want to get some "real-world" thoughts from those of you who have done this stuff for a while.

16 Comments ••• 91% Upvoted

Sort By: Best

· 3 yr. ago

I'd recommend you to use Helm: https://helm.sh

You can create Helm charts per microservice which will allow you to have defined in one place all the configuration that the microservice will need. You could even reuse those charts if you're ever going to run those microservice in another project.

It'll also let you do easy releases through your CI/CD pipeline.

Hope this helps!

16 Reply Share •••

· 3 yr. ago

I'd advise against using helm like this. Helm can be useful when a service is packaged to be run by someone who isn't familiar with the service, and just wants to paste in a few values. This doesn't sound like the problem you've identified, so I'm not sure if Helm provides value to you right now.

To help answer your question, how is your code stored? Do you have a single repo with all of these services, or does each microservice live in a separate repo? And how do you deploy? What tools are you using to deploy after build?

1 Reply Share •••

Continue this thread →

· 3 yr. ago

Helm is generally considered bad practice when it comes to k8s.

-2 Reply Share ...

Continue this thread →

· 3 yr. ago

GitOps; check them all into a git repo, prohibit users from direct changes to master, allow merges to master with approval of someone else with commit rights && tests passed, have something track master's HEAD and continually apply it to the cluster.

Even if someone pushes through a change that shouldn't have happened, you have an immutable audit trail and a revert button.

Just need rigor to avoid commit messages like "Stuff". Possibly enforce reference to a bug/story ticket, or a test that refuses to pass for short commit messages, or coworkers that refuse to accept uncommented mystery changes.

7 Reply Share ...

· 3 yr. ago

I am glad you said this. I watched this bit on Jenkins-X and was trying to understand the flow of this.

If I understand right, GitOps uses one repo PER environment. In each repo is essentially the same/similar sit of config files (e.g one (or two?) per microservice/component), configured for the environment. E.g. production might be set with an ingress configuration for GKE deployment, more replicas of each service (some more than others maybe), etc, where as a DEV or integration ENV might not need replicas.. it just wants to ensure the thing builds and passes unit/integration tests. Is that right?

It also assumes, if I understand it right, that you essentially use MASTER on each of your microservice/etc repos, to kick off the CI/CD cycle. So if I were to be fixing a bug or adding a new feature, I create a branch off of master, I work in that, then I submit a PR. Once the PR is approved and there are no conflicts, I merge this in to master (of my service repo). CI/CD is configured to "watch" all these service repos, and upon a push to the master branch, kicks off the CI/CD process, which would hopefully yield a company/central docker image repo, and ONLY build the repos that changed.. e.g. compile, run unit tests, then build docker images and push those images to the repo. The next step is to then (assuming it is set up), spin up an environment for something like automated tests. This would then pull in all the latest images, run them and run tests. If this passes, the next phase is to either do another ENV cycle.. OR it would then (I think this is where GitOps comes in) commit/push a change to a given GitOps repo?

OR maybe I got that wrong.. still a little confused on how you set all this up. I understand you can use Jenkins and scripts, etc. But I also though Jenkins-X basically was a CI/CD product that you deploy and then use a UI to set this stuff up. No?

3 Reply Share ...

Continue this thread →

· 3 yr. ago

There's definitely tons of options. We templatise the kube artifacts via Helm and substitute values in through our ci/CD. When connecting a bunch of services, I've kept the helm charts inside the repository of the applications, just in a "deploy" or "helm" dir.

Then all of the deployment scripts for the pipeline live in a separate git repo. The pipeline manages each component individually, and a change to one component triggers a deploy to only the components that were changed.

5 Reply Share ...

· 3 yr. ago

We have too many options and none of them are really good.

1 Reply Share ...

· 3 yr. ago

This is a fantastic question, and one that will come up again and again on teams if you don't have some kind of standard. That being said, you have to start somewhere. I started by having a config directory and then namespace/environment specific directories under config to hold specific manifest files for things like configmaps, deployments, ingresses, services, etc

services, etc.

Something similar to this:

```
└── config
    ├── dev
    │   ├── configmaps
    │   ├── deployments
    │   ├── ingresses
    │   └── services
    ├── prd
    │   ├── configmaps
    │   ├── deployments
    │   ├── ingresses
    │   └── services
    ├── qat
    │   ├── configmaps
    │   ├── deployments
    │   ├── ingresses
    │   └── services
    └── stg
        ├── configmaps
        ├── deployments
        ├── ingresses
        └── services
```

That was kind of revision 1. This works really well, and you can setup a pretty nice CI pipeline around this with some simple bash scripts, though like /u/davilag mentioned, helm is the way to go. This is what we've started using for most everything. Once you get in there and start messing around with it, you'll find that it's really not that complex, especially if you're already using k8s in your day-to-day.

EDIT: Also to /u/alanjcastonguay 's point, you should 100% be getting into a gitops mindframe. Check out ArgoCD or Weaveworks Flux (argo is really cool) for tools that can help you get started on that path.

TL;DR - Structured Repos and Helm. Put in the time to learn helm and you won't be sorry

5 Reply Share ...

· 3 yr. ago

Sorry stupid question, coming from someone just looking at k8s...

Wouldn't the idea be to reuse each of your files with variable substitution? E.g. you servicedefinition or configmaps would be the same keys across environments, just different values?

So with your above layout, if you need to add a new service or modify an existing service, you'd have to do that work 4x times or whatever.

2 Reply Share ...