aws

COMMUNITY DAY

Microservices are ‘easy’ dependencies are hard: The right way to build a cloud-native CI/CD

Itiel Shwartz

aws

COMMUNITY DAY

About me

- Joined Rookout as a first developer and production engineer
- Previously Worked at Forter and eBay as a backend engineer
- @itielshwartz on both Github and Twitter

personal blog at: https://etlsh.com

Why do I need Kubernetes and what can it do?

Kubernetes has a number of features. It can be thought of as:

- a container platform
- a microservices platform
- a portable cloud platform and a lot more.[1]

[1] https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

So is this talk going to be -
why K8S and helm are so awesome?

HELL NO!

aws

COMMUNITY DAY

Kubernetes is great, BUT...

- People tend to abuse it
- We are going to talk about moving fast, while staying alive.
- By the end of this session, you will be more proficient on how to use K8S

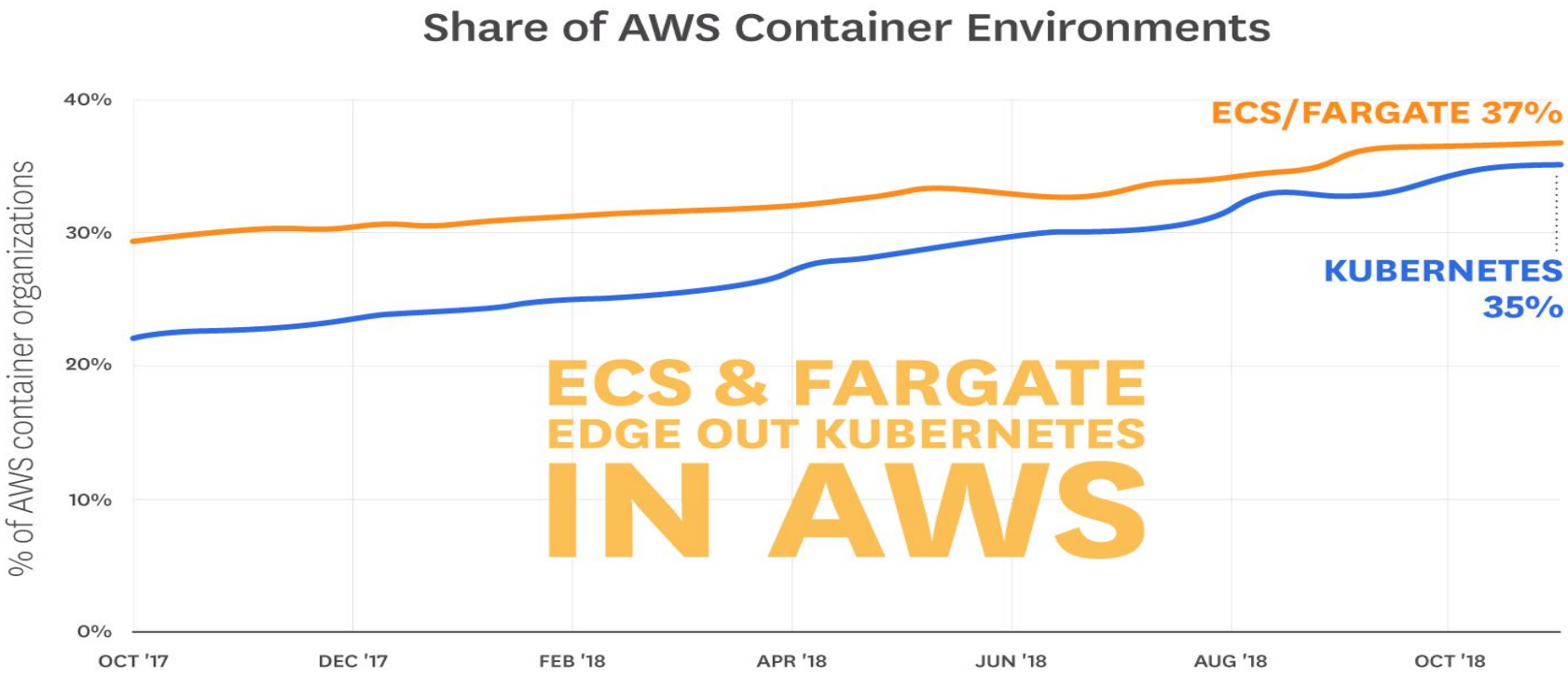
aws
COMMUNITY DAY

EKS VS ECS VS Fargate?

Amazon ECS

Amazon EKS

AWS Fargate



Source: Datadog

aws
COMMUNITY DAY

Agenda

1. Build a basic app
2. K8S it
3. See why K8S isn’t enough
4. Helm it
5. Little Jenkins CI/CD
6. Summary

The monolith

aws

COMMUNITY DAY

Our app

```python
import json
import os
import random

from flask import Flask, jsonify

app = Flask(__name__)
DIR_PATH = os.path.dirname(os.path.realpath(__file__))
QUOTES_FILE_PATH = os.path.join(DIR_PATH, "quotes.json")

with open(QUOTES_FILE_PATH) as f:
    # Quote list
    quotes = json.load(f)


@app.route("/")
def get_quote():
    """
    :return: return a random quote
    """
    random_number = random.randint(0, len(quotes) - 1)
    return jsonify(quotes[random_number])


if __name__ == "__main__":
    app.run("0.0.0.0")
```

Response example

```
{
  "author": "Unknown",
  "text": "It works on my machine."
}
```

aws
COMMUNITY DAY

Our dockered app

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers*

Dockerfile

```
FROM python:3.7-slim

WORKDIR /usr/src/index

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python", "./index.py" ]
```

aws

COMMUNITY DAY

Our K8S app

A Kubernetes pod is a group of containers that are deployed together on the same host. If you frequently deploy single containers, you can generally replace the word "pod" with "container" and accurately understand the concept.

A Deployment controller provides declarative updates for Pods and ReplicaSets

Deployment.yaml

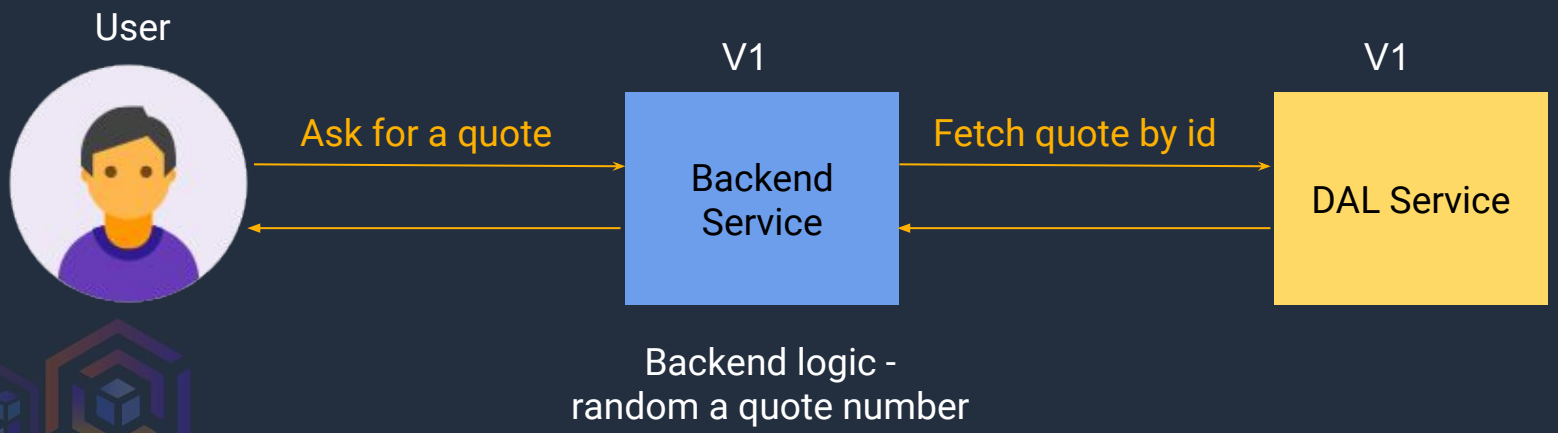
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: main
  labels:
    name: main
spec:
  replicas: 1
  selector:
    matchLabels:
      name: main
  template:
    metadata:
      labels:
        name: main
    spec:
      containers:
      - name: main
        image: itielshwartz/microservice-using-helm:part-3-with-k8s
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
```

K8S is so easy!

Breaking the monolith

aws
COMMUNITY DAY

Going to micro service - Architecture



aws

COMMUNITY DAY

Going to micro service - CODE

Backend

```
app = Flask(__name__)
URL_FOR_QUOTE_DB_WRAPPER = os.environ.get("DAL_URL", "http://localhost:5000")


# We get the number of quotes the DB have
NUMBER_OF_QUOTES = int(requests.get(URL_FOR_QUOTE_DB_WRAPPER + "/quote_len").text)


@app.route("/")
def get_quote():
    """
    :return: return a random quote
    """
    random_number = random.randint(0, NUMBER_OF_QUOTES - 1)
    url_for_quote_request = "/quote/{}".format(random_number)
    return jsonify(
        requests.get(URL_FOR_QUOTE_DB_WRAPPER + url_for_quote_request).json()
    )
```

DAL

```
DIR_PATH = os.path.dirname(os.path.realpath(__file__))
QUOTES_FILE_PATH = os.path.join(DIR_PATH, "quotes.json")

with open(QUOTES_FILE_PATH) as f:
    quotes = json.load(f)


@app.route("/quote_len")
def get_quote_len():
    return str(len(quotes))


@app.route("/quote/<quote_number>")
def get_quote(quote_number):
    return jsonify(quotes[int(quote_number)])
```

aws

COMMUNITY DAY

Customers

aws

COMMUNITY DAY

Adding search to our microservice

DAL

```
@app.route("/search")
def search_quote():
    searchword = request.args.get("q", "")
    print("Searching for {}".format(searchword))
    for quote in quotes:
        if searchword.lower() in quote["text"]:
            return jsonify(quote)
    print("No result found - default the first quote")
    return jsonify(quotes[0])
```

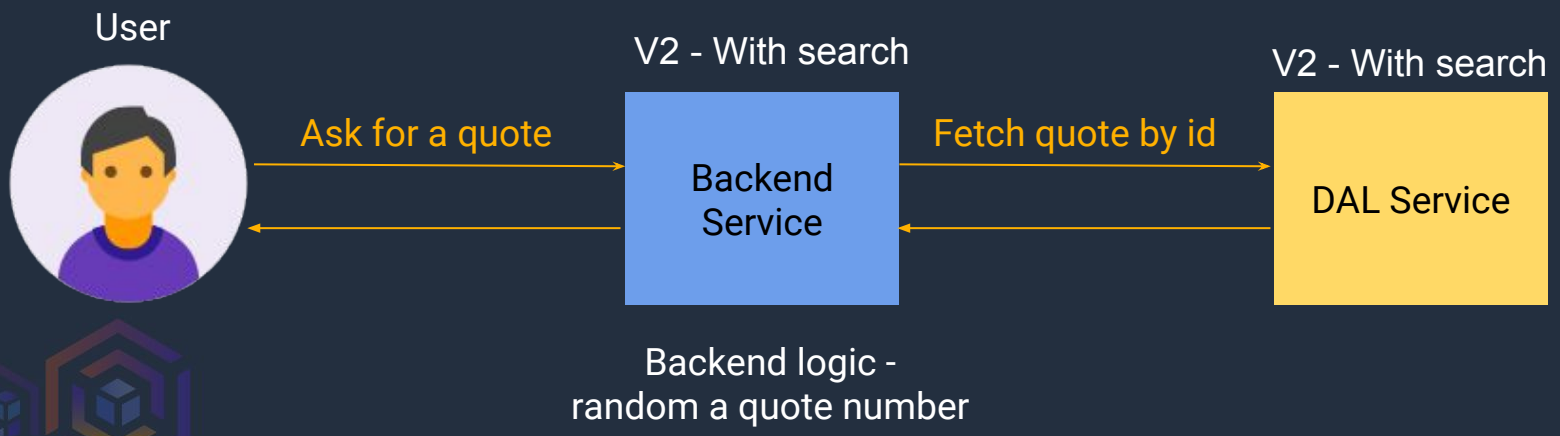
backend

```
@app.route("/search")
def search_quote():
    searchword = request.args.get("q", "")
    return jsonify(
        requests.get(
            URL_FOR_QUOTE_DB_WRAPPER + "/search", params={"q": searchword}
        ).json()
    )
```

Now we can **get** and **search** quotes :)

aws
COMMUNITY DAY

Going to micro service - Architecture




let’s not panic

aws

COMMUNITY DAY

My bad, I added another small function to this deploy

DAL

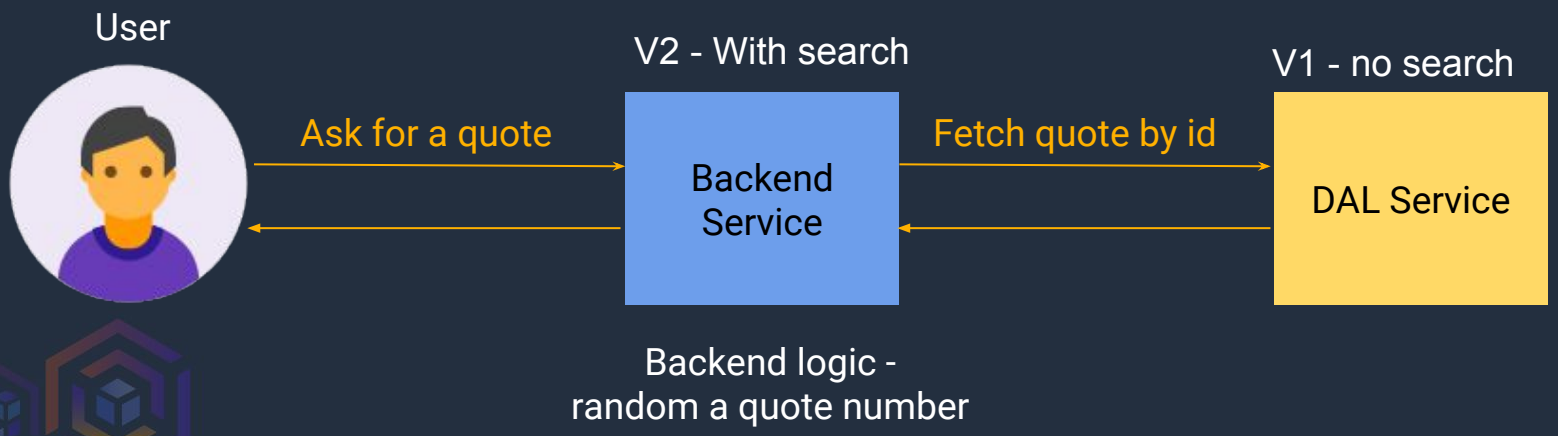
```
@app.route("/quote/<quote_number>")
def get_quote(quote_number):
    sanity_check()
    return jsonify(quotes[int(quote_number)])


def sanity_check():
    if "darwin" in platform:
        print("All is good")
    else:
        print("You should always run on mac, this code should never happen")
        exit(1)
```

All I need is to revert the DAL

Phew... this was close

aws
COMMUNITY DAY

Going to micro service - Architecture

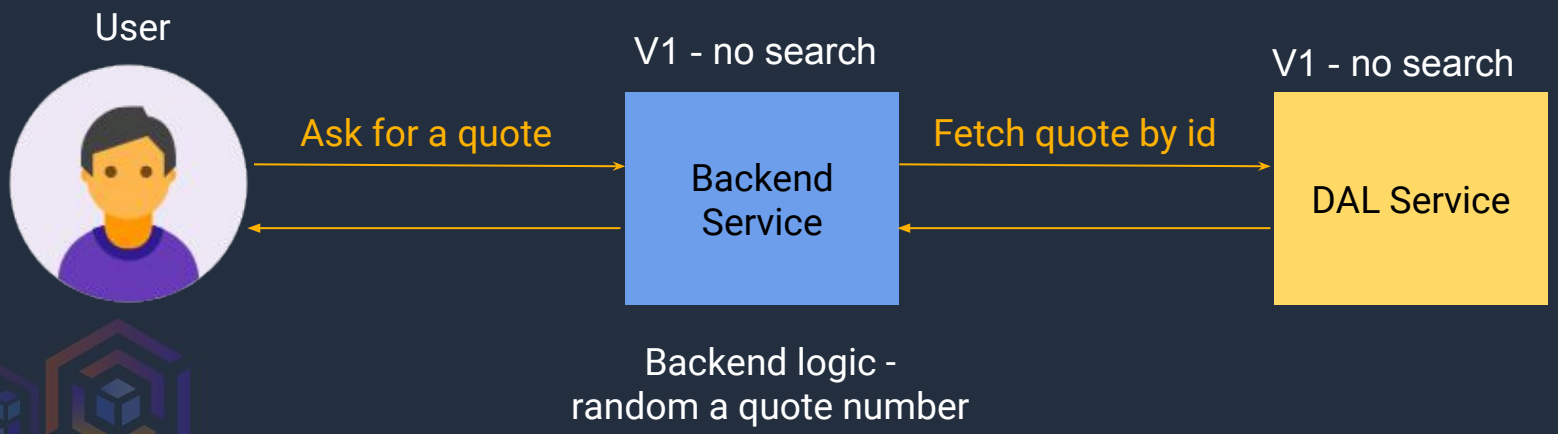


Ok, let’s not panic

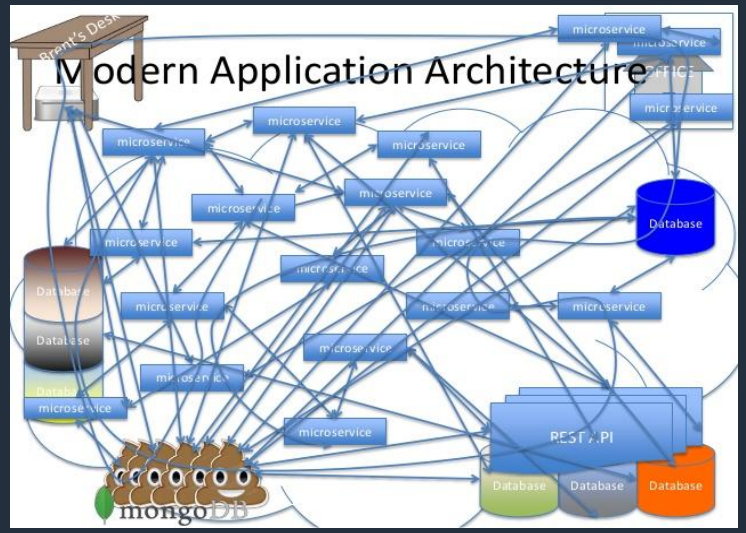
All I need is to revert everything...

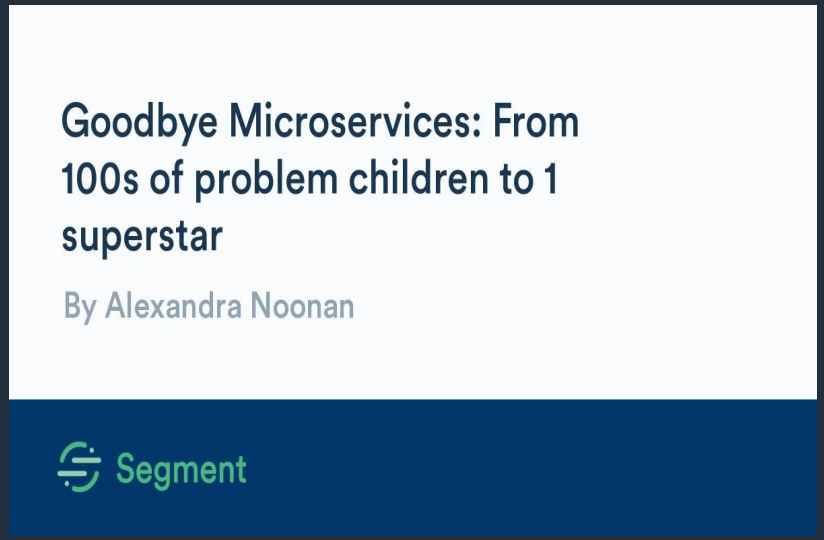
aws
COMMUNITY DAY

Going to micro service - Architecture



I thought microservice are the best!





aws

COMMUNITY DAY

The problems we faced in “vanilla” k8s

1. No version control for the dependencies between microservices!!!
2. We write a lot of boilerplate for each microservice
   - Deployment.yaml , service.yaml, LB.yaml , configmap.yaml
3. No easy way testing those microservices before prod
   - we need to configure each env

aws

COMMUNITY DAY

Move fast and break things. And then what?

1. The ability to deploy new code to production fast - everyone is talking about it!
2. The ability to test the system e2e very fast before going to production
3. Fast rollback - Your system will break! Not a question of if, but a question of when.

Helm to the rescue

aws

COMMUNITY DAY

What is helm?

Helm describes itself as the “The package manager for Kubernetes”

sounds pretty straightforward right? No? Maybe? Does it help you to Download packages? Upload packages? Deploy envs? State control? Cluster control? Templating?

aws

COMMUNITY DAY

Helm Glossary

CHART

REPOSITORY (REPO, CHART REPOSITORY)

RELEASE

aws

COMMUNITY DAY

Chart? Chart!

A chart is organized as a collection of files inside of a directory.

The directory name is the name of the chart (without versioning information). Thus, a chart describing WordPress would be stored in the wordpress/ directory*

```
wordpress/
  Chart.yaml          # A YAML file containing information about the chart
  LICENSE             # OPTIONAL: A plain text file containing the license for the chart
  README.md           # OPTIONAL: A human-readable README file
  requirements.yaml   # OPTIONAL: A YAML file listing dependencies for the chart
  values.yaml         # The default configuration values for this chart
  charts/             # A directory containing any charts upon which this chart depends.
  templates/          # A directory of templates that, when combined with values,
                      # will generate valid Kubernetes manifest files.
  templates/NOTES.txt # OPTIONAL: A plain text file containing short usage notes
```

aws

COMMUNITY DAY

Chart - Sounds hard? Not really

chart.yaml

```yaml
apiVersion: v1
description: Chart for the backend logic of our quotes service
name: quotes-backend-chart
version: 1.0.0
```

values.yaml

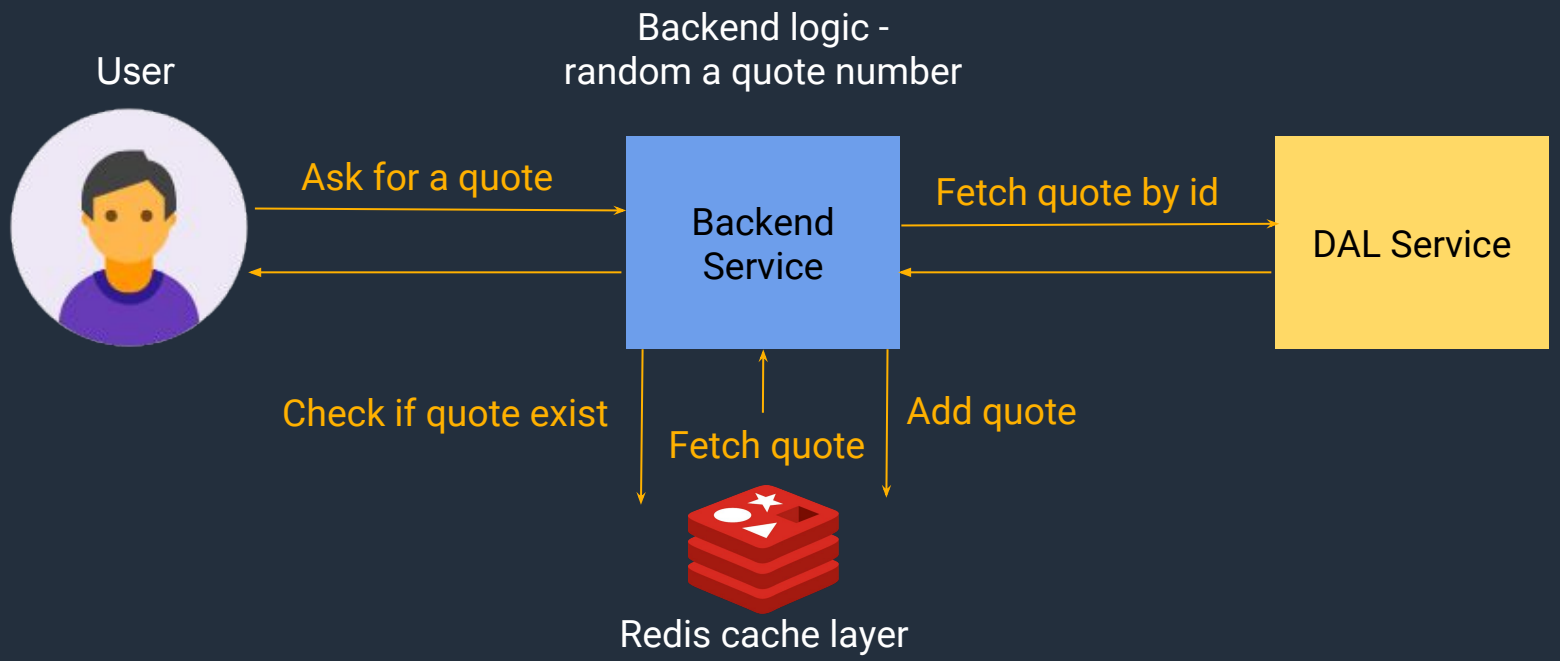
```yaml
dockerTag: itielshwartz/microservice-using-helm:part-4-backend
```

deployment.yaml

```yaml
image: {{ .Values.dockerTag }}
```

aws
COMMUNITY DAY

Adding redis to our infra



aws

COMMUNITY DAY

Using helm requirements.yaml

The requirements file is one of the most “overlooked” features helm has to offer, and one of its greatest!

It allows us to:

- Add external dependencies
- Deploy multiple services in “one-shot”

Redis helm

Requirement.yaml

```yaml
dependencies:
- name: redis
  version: "4.2.5"
  repository: '@stable'
```

Umbrella chart

aws

COMMUNITY DAY

One chart to rule them all

We created a single file with all microservices, and to specify a version for each one.

- We can use a remote repository.

umbrella/requirements.yaml

```yaml
dependencies:
- name: dal-chart
  version: "1.0.0"
  repository: "file://../db_service_wrapper/dal-chart"
- name: quotes-backend-chart
  version: "1.0.0"
  repository: "file://../backend/quotes-backend-chart"
```

aws

COMMUNITY DAY

Going full on Helm

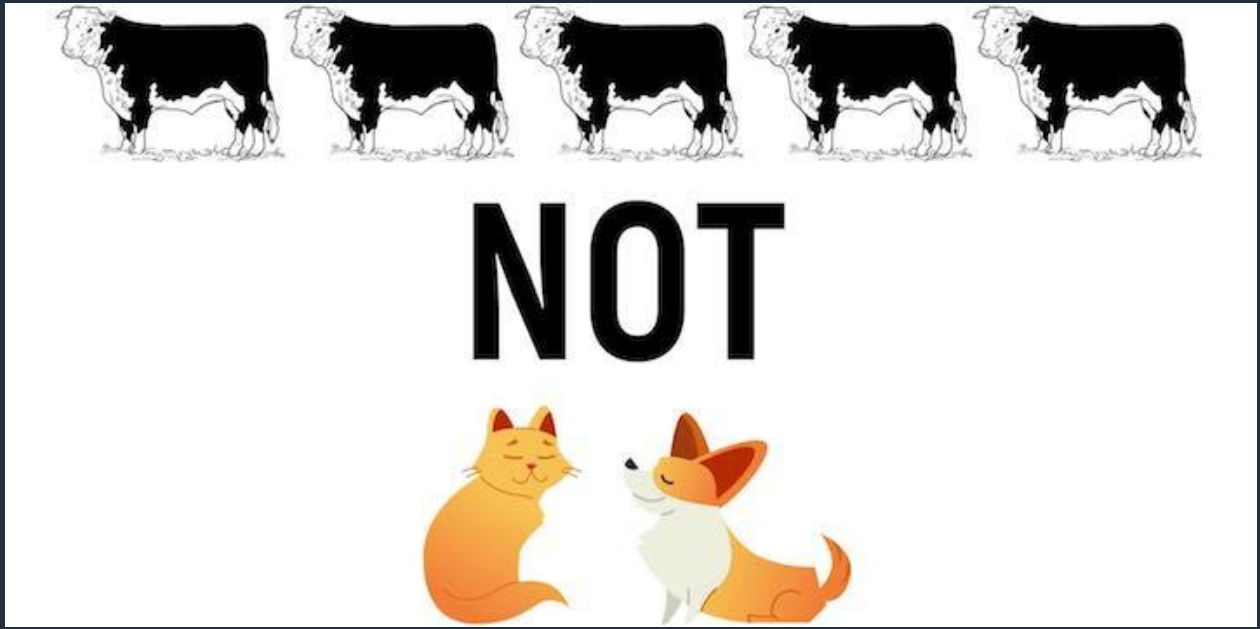
The power of a single helm chart for deployment:

1. We now have a single-source of truth about our microservices versions.
2. We can rollback to a previous deployment (and by that revert all changes in all microservices) . (helm rollback X)
3. We can Deploy to a new namespace! Meaning adding prod env and staging env using the same files. (helm --namespace prod)
4. We can inject different variable based on the env we are deploying to (using the values.yaml files) (helm -f prod.yaml)

aws

COMMUNITY DAY

Treat your CI/CD like cattle, not pets!



aws

COMMUNITY DAY

CI/CD blueprint for helm and K8S

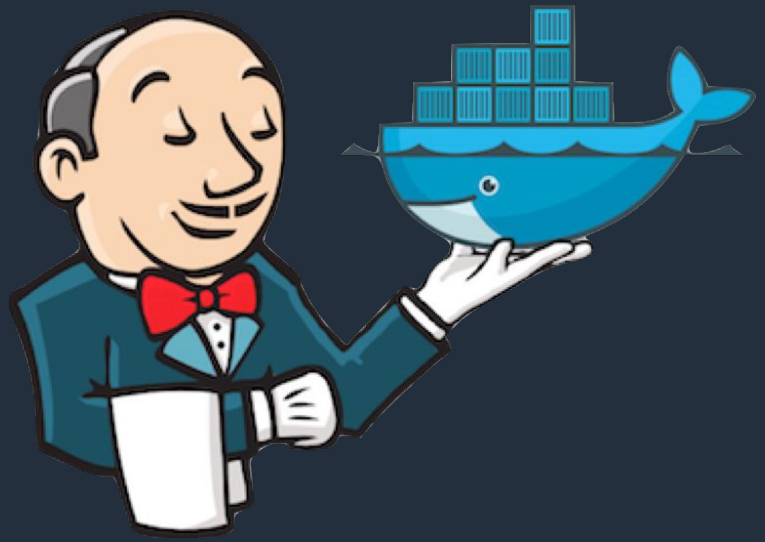
Full helm microservice multiple repo ci/cd pipeline example:

1. Push to branch.
2. Build the docker.
3. Bonus: Use multistage docker build to run tests
4. Build the helm package.

   If building master:

5. Try and update the umbrella chart (update backend from 1.0.x -> 1.0.x+1)
6. Deploy to staging/dev namespace
7. If it works - great! (and maybe auto move to prod)
8. If it fails -> revert the umbrella chart change (and maybe revert the backend branch)

Death to Jenkins!



Long live Jenkins!

aws

COMMUNITY DAY

Jenkins CI/CD, Helm and K8S

1. With Jenkins shared-libraries we can create a full pipeline that will be shared across all repos.
2. When adding a new micro service we don't need to write a single line of code!
3. This also allow us to config all of our CI/CD as code (Groovy).
4. Using Jenkins and helm we can create a full CI/CD pipeline that is standard across all microservices.
5. Jenkins has it own Helm chart - little meta but works :)
6. On the other hand - Jenkins.

Jenkinsfile

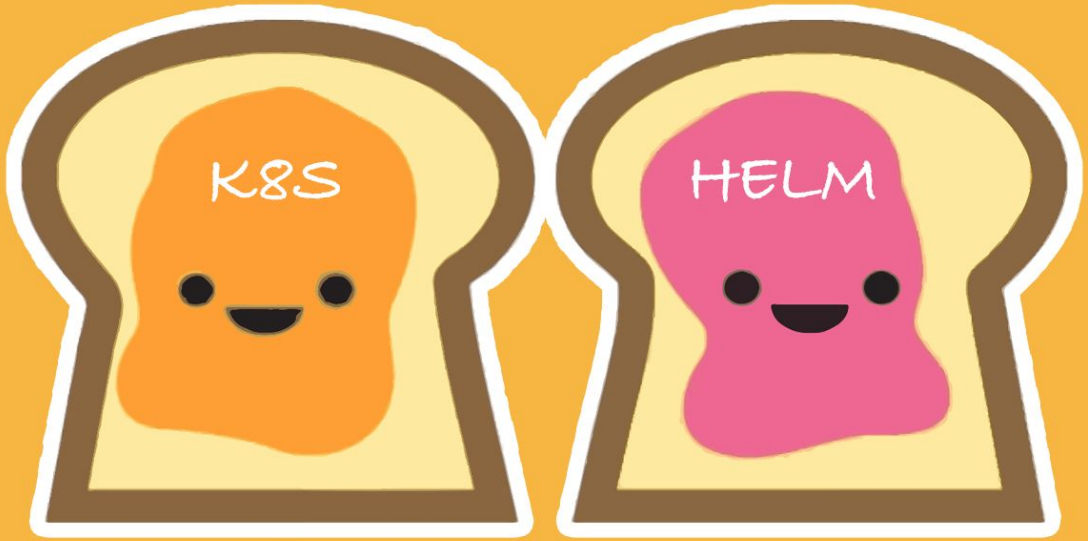
```
basicPipeline {
    PROFILE = "MicroService"
    COMPONENT = "backend"
}
```

Summary

1. Managing microservice is hard.
2. Helm can make it easier.
3. All source code can be found on my github:

   https://github.com/itielshwartz/microservices-using-helm

We live in a cool time, where the problem is not deploying - but deploying too fast :)



Like peanut butter and jelly