Run, Create, and Expose Generators

- These commands use helper templates called "generators"
- Every resource in Kubernetes has a specification or "spec"

```
> kubectl create deployment sample --image nginx --dry-run -o yaml
```

- You can output those templates with `--dry-run -o yaml`
- You can use those YAML defaults as a starting point
- Generators are "opinionated defaults"

Generator Examples

- Using dry-run with yaml output we can see the generators

```
> kubectl create deployment test --image nginx --dry-run -o yaml
> kubectl create job test --image nginx --dry-run -o yaml
> kubectl expose deployment/test --port 80 --dry-run -o yaml
```

  - You need the deployment to exist before this works

Cleanup

- Let's remove the Deployment

```
> kubectl delete deployment test
```

The Future of kubectl run

- Right now (1.12-1.15) run is in a state of flux
- The goal is to reduce its features to only create Pods
  - Right now it defaults to creating Deployments (with the warning)
  - It has lots of generators but they are all deprecated
  - The idea is to make it easy like docker run for one-off tasks
- It's not recommended for production
- Use for simple dev/test or troubleshooting pods

Old Run Confusion

- The generators activate different Controllers based on options
- Using dry-run we can see which generators are used

```
> kubectl run test --image nginx --dry-run
> kubectl run test --image nginx --port 80 --expose --dry-run
> kubectl run test --image nginx --restart OnFailure --dry-run
> kubectl run test --image nginx --restart Never --
  dry-run
> kubectl run test --image nginx --schedule "*/1 *
  * * *" --dry-run
```

Imperative vs. Declarative

- Imperative: Focus on *how* a program operates
- Declarative: Focus on *what* a program should accomplish
- Example: "I'd like a cup of coffee"
- Imperative: I boil water, scoop out 42 grams of medium-fine grounds, poor over 700 grams of water, etc.
- Declarative: "Barista, I'd like a a cup of coffee". (Barista is the engine that works through the steps, including retrying to make a cup, and is only finished when I have a cup)

Kubernetes Imperative

- Examples: kubectl run, kubectl create deployment, kubectl update
  - We start with a state we know (no deployment exists)
  - We ask kubectl run to create a deployment
- Different commands are required to change that deployment
- Different commands are required per object
- Imperative is easier when you know the state
- Imperative is easier to get started
- Imperative is easier for humans at the CLI
- Imperative is NOT easy to automate

Kubernetes Declarative

- Example: kubectl apply -f my-resources.yaml
  - We don't know the current state
  - We only know what we want the end result to be (yaml contents)
- Same command each time (tiny exception for delete)
- Resources can be all in a file, or many files (apply a whole dir)
- Requires understanding the YAML keys and values
- More work than kubectl run for just starting a pod
- The easiest way to automate
- The eventual path to GitOps happiness

Three Management Approaches

- **Imperative commands:** run, expose, scale, edit, create deployment
  - Best for dev/learning/personal projects
  - Easy to learn, hardest to manage over time
- **Imperative objects:** create -f file.yml, replace -f file.yml, delete…
  - Good for prod of small environments, single file per command
  - Store your changes in git-based yaml files
  - Hard to automate
- **Declarative objects:** apply -f file.yml or dir\, diff
  - Best for prod, easier to automate
  - Harder to understand and predict changes

Three Management Approaches

- **Most Important Rule:**
  - Don't mix the three approaches
- **Bret's recommendations:**
  - Learn the Imperative CLI for easy control of local and test setups
  - Move to apply -f file.yml and apply -f directory\ for prod
  - Store yaml in git, git commit each change before you apply
  - This trains you for later doing GitOps (where git commits are automatically applied to clusters)