Kui

CLIs with a GUI Twist

Nick Mitchell **IBM Research** **September 2021**

https://github.com/kubernetes-sigs/kui

We Love CLIs

But they’re kind of stuck in the 70s. Can we preserve what we love about them, while bringing in well-deserved graphical enhancements?

The Kui Approach

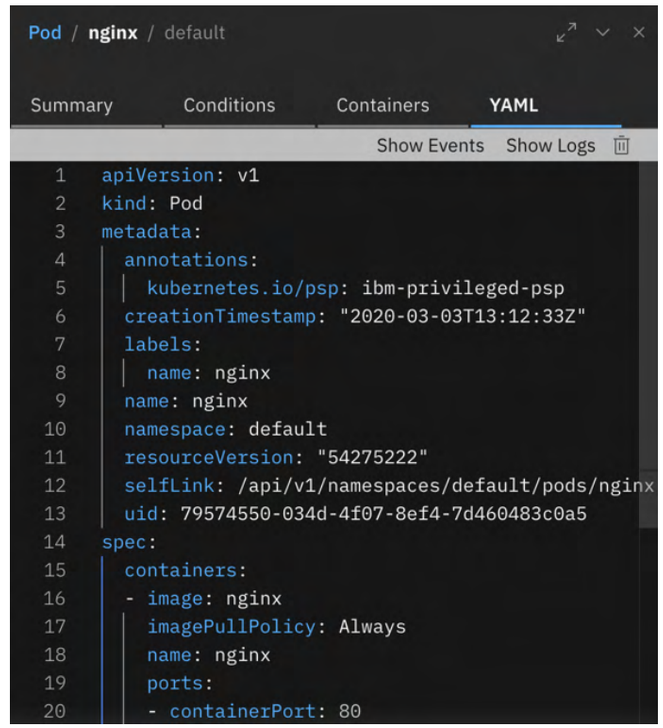
Kui enhances four aspects of CLIs

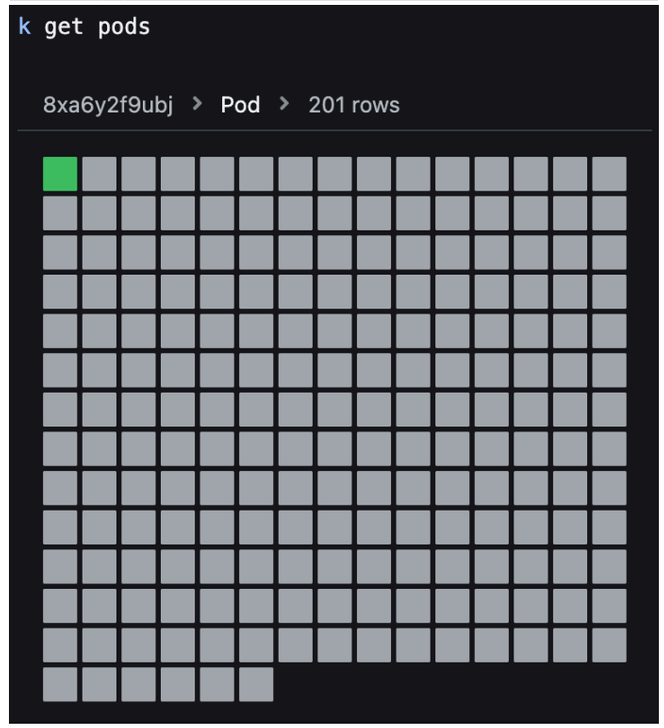
1. CLI commands produce graphics

2. Context info is graphical and interactive
   - Not limited to ASCII art like Oh My Zsh
   - e.g. display and **change** Kube context or namespace

3. **Hyperlinks are powered up**
   - Not limited to URLs, as in ASCII terminals
   - e.g. click on table row to drill down to the resource details

4. **Terminal splits integrated into navigation**
   - target=_split
   - e.g. clicking on a table row can open up the drilldown in a separate split





Agenda

Kui as a **Tool** you can use today

Kui as a **Framework** for enhancing any CLI

Kui as a **Notebook** sharing mechanism

Part 1: Kui as a Kubernetes Tool

Kui

Kui | Tab 1

```
k get pods                                                    4.4s
```

8xa6y2f9ubj > Pod > 201 rows

Name	Status	Restarts
kui-00001-deployment-5bcd6bc9b-zshvn		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-1		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-10		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-100		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-11		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-12		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-13		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-14		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-15		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-16		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-17		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-18		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-19		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-2		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-20		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-21		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-22		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-23		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-24		0
super-08867868-3bc3-4e21-aaee-dbc1c563f838-25		0

```
kubectl get Pod kui-00001-deployment-5bcd6bc9b-zshvn -o yaml -n 8xa6y2f9ubj    350ms
```

Pod > kui-00001 > deployment-5bcd6bc9b-zshvn > 8xa6y2f9ubj

Summary | Logs | Terminal | YAML

Created on **9/8/2021, 5:46:49 PM**. Version **1554405993**.

```
1  Name: kui-00001-deployment-5bcd6bc9b-zshvn
2  Ready: 3/3
3  Status: Running
4  Restarts: 0
5  Age: 15h 59m 18.5s
6  IP: 172.30.72.154
7  Node: 10.240.64.77
8  Nominated node: <none>
9  Readiness gates: <none>
10
```

Show Events | Show Owner Reference | Show Node

```
kubectl get Pod super-08867868-3bc3-4e21-aaee-dbc1c563f838-11 -o yaml -n 8xa6y2f9ubj    197ms
```

Pod > super-08867868-3bc3-4e21-aaee-dbc1c563f838-11 > 8xa6y2f9ubj

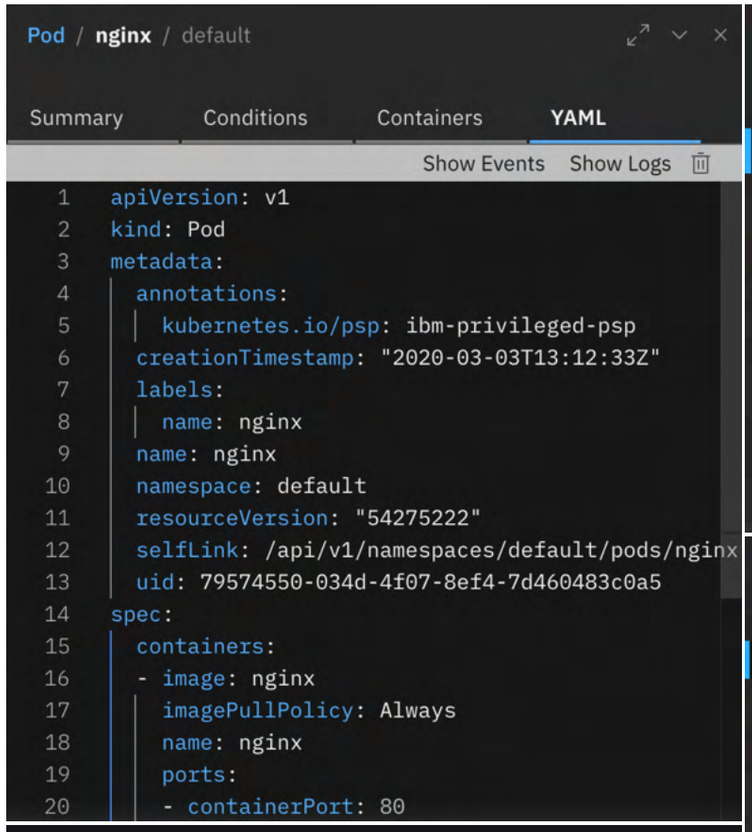
Summary | Logs | Terminal | YAML

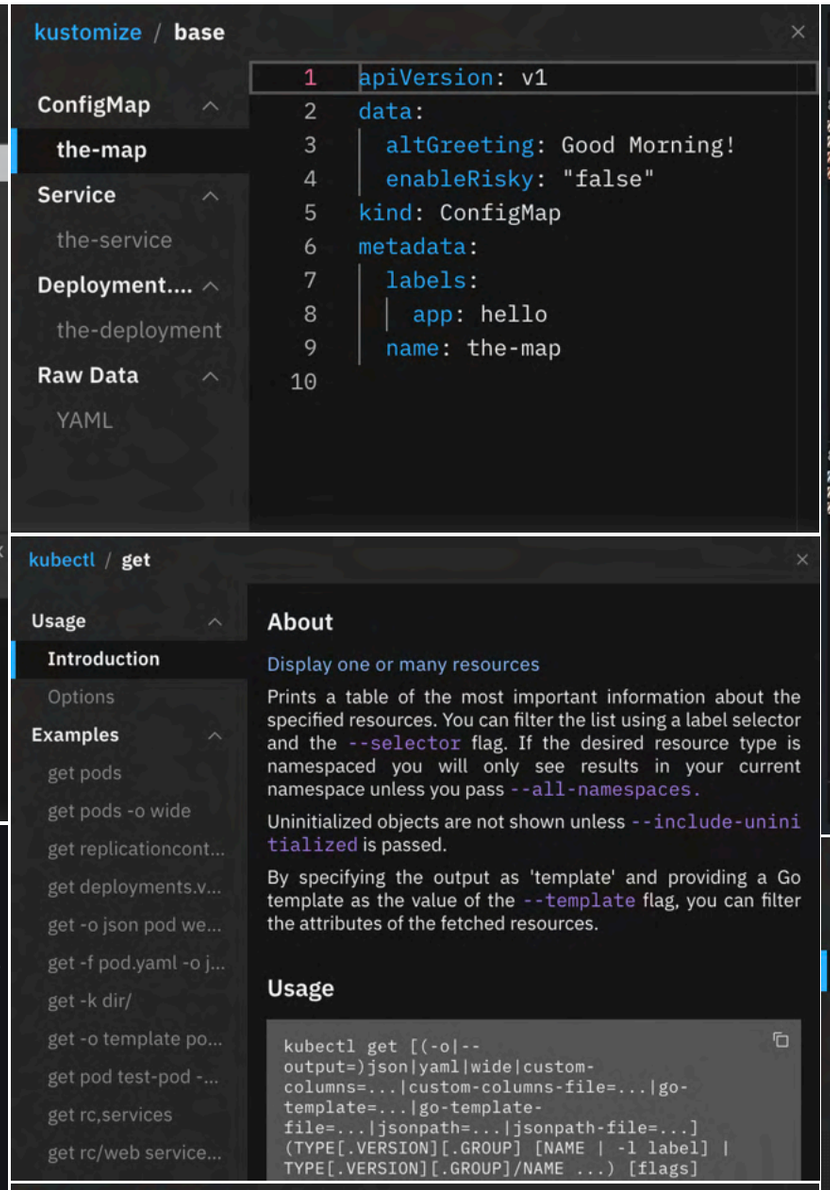
Created on **9/9/2021, 6:30:16 AM**. Version **1558211389**.

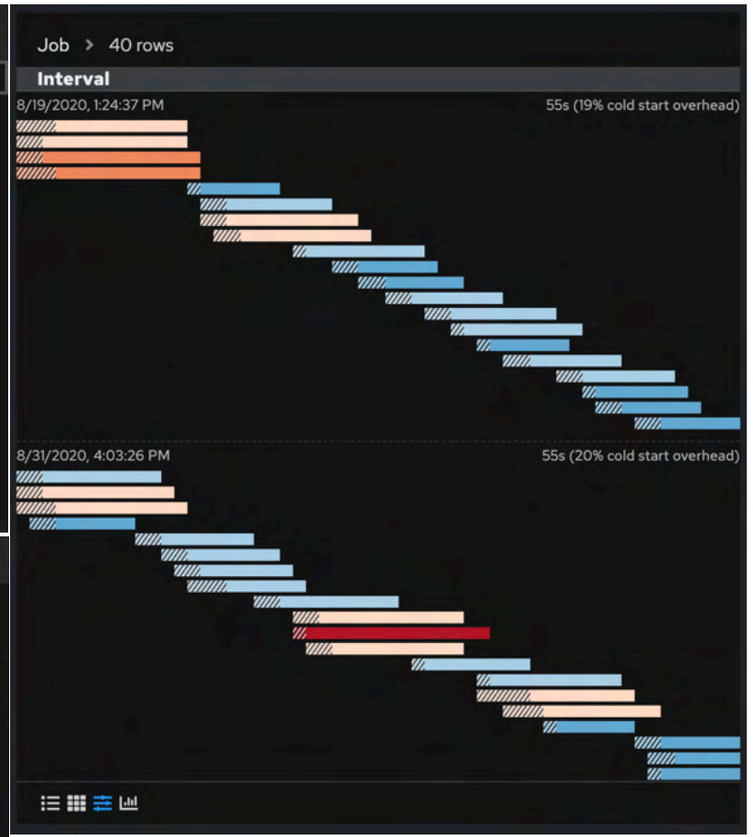
```
1  Name: super-08867868-3bc3-4e21-aaee-dbc1c563f838-11
2  Ready: 0/1
3  Status: Succeeded
4  Restarts: 0
5  Age: 3h 16m 23.5s
```

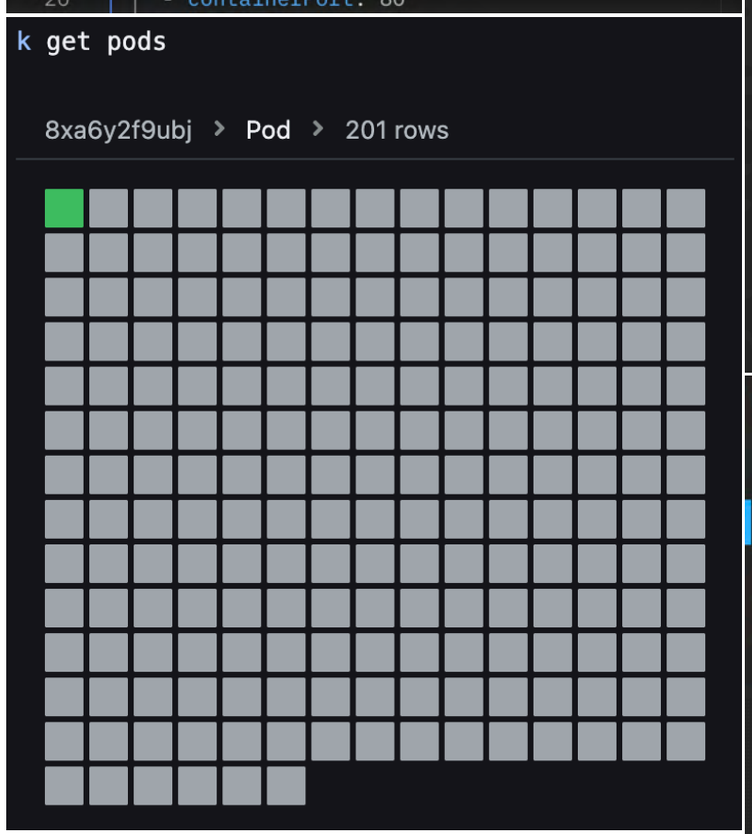
/Users/nickm | 8xa6y2f9ubj | PatternFly4 Light

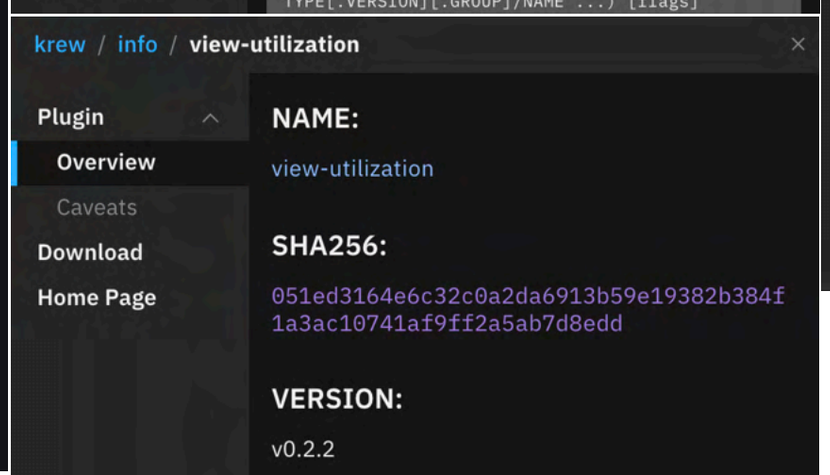
Features of Kui

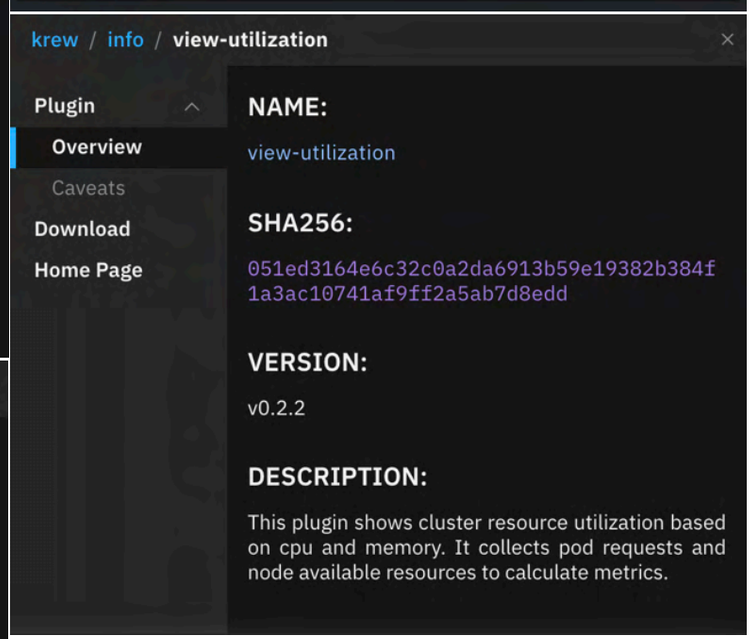












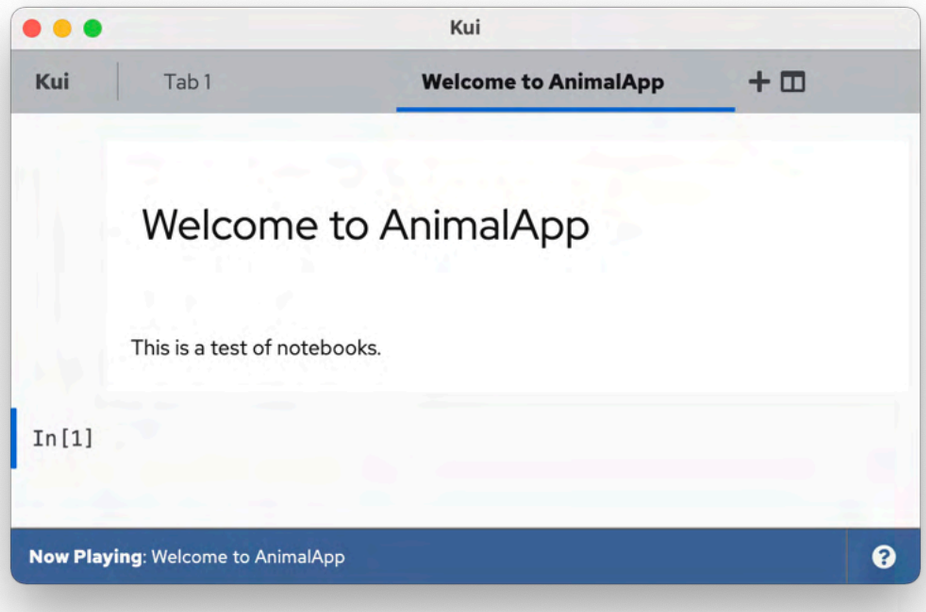
default > Pod > 56 rows

```
[0s] pod/nginx: Successfully pulled image "nginx"
[0s] pod/nginx: Created container nginx
[0s] pod/nginx: Started container nginx
[0s] pod/nginx2: Stopping container nginx
[0s] pod/nginx: Stopping container nginx
[0s] pod/kui-crashy: Back-off restarting failed container
```

Part 2: Kui as a Framework for Enhancing CLIs

Getting Started with your Customizations

- To enhance Kui, or use a custom theme and icon, you can start from the template repository
- Enhancements involve writing TypeScript code
- **Fork the template!**
  - https://github.com/kui-shell/AnimalApp/generate
  - git clone ...
  - npm ci
  - npm run watch
  - npm run open



Kui Framework: The General Approach

Three Options for Enhancement

1. **Listen and override** executions of chosen subcommands
   - e.g. "I want `kubectl get pods` to return something totally custom"
   - The return value of your command handler is your desired graphical response

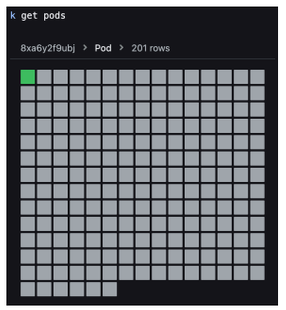
2. **Decorate** the default view for chosen Kube resource kinds
   - e.g. "I want to show a fancy visualization for logs of my pods"
   - Your view will show up as a tab when viewing instances of those kinds

3. Add a context **widget**
   - e.g. "I want to allow quick selection of region target for my Cloud provider"

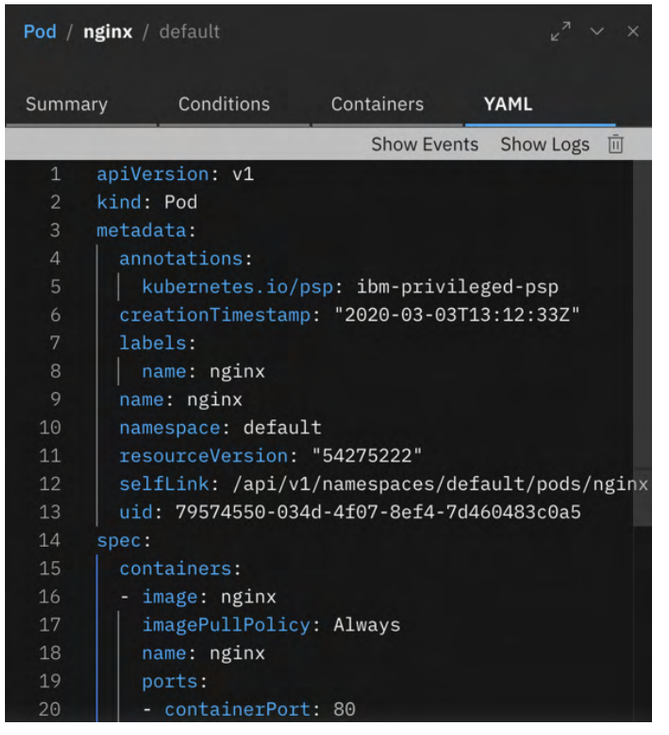
Supported Graphical Models

For fully custom overrides of CLI subcommands

- Tabular views
  - **Table** with clickable rows, live status updates, a stream of relevant events
  - **Grid** to allow for dense packing of lots of table rows
  - **Sequence Diagram** for jobs
  - **Source links** to offer one-click connections between CRUD operations and source code



- Syntax-colored source
- Multi-tab view
  - Useful for presenting resource details
- **React** component
  - You can prototype additions to core Kui views in this way



Part 3: Kui as a Notebook method for sharing and learning

Kui

Kui | Tab 1 | Iter8 Tutorial 1

Iter8 Tutorial 1: Verifying Performance and Error Rate Quality

In this short demo we will walkthrough an **application upgrade scenario**. We would like to promote a *candidate* version to production environment.

Follow the **Activity List** below to walk through this scenario.

Activity List

- Deploy the baseline and candidate applications
- Start the Iter8 Experiment
- Monitor the Progress of the Experiment

Use Case Details

Your application is deployed to a production Kubernetes environment. Subequent development has resulted in a new version of the application. It has passed its unit and functional test cases. Is it ready to deploy to production?

Experience shows, however, that new

This demo assumes that you are running in a k8s cluster with no istio support and the way you carry out your upgrades is by leveraging `kubectl apply` which will progressively promote one version over the other. In case you are leverage `istio` or `knative` demos `1` and `2` show how to leverage `iter8` in combination with `istio` and `knative` to carry out upgrades.

Step 1 Deploy the baseline and candidate applications

First, we use `kubectl apply` to deploy the two versions of our sample application.

```
[1] k apply -f https://raw.githubusercontent.com/kalantar/iter8/cil/samples/cil/first-exp/hello-candidate.yaml
```

Show hello-candidate.yaml

Show Sample Output

Step 2 Start the Iter8 Experiment

Next, we would like to promote `hello-candidate` to production, provided that it satisfies a series of SLOs(service level objectives). In this case we are focussing on the metrics below, but these can be any in-built or *user-definied* metrics.

1. Mean latency rate for the application should be > x
2. Median latency rate for the application should be > x

Now Playing: Iter8 Tutorial 1 | PatternFly4 Light

Kui

Kui | Tab 1 | Kubernetes – Deploying Applications

Deploying PHP Guestbook Application with Redis

This tutorial shows you how to build and deploy a simple, multi-tier web application using Kubernetes and Kui. You will learn how to **explore the structure** of this application. Once you are confident in your understanding of the implications of its deployment, this notebook can help you with taking that next step.

The guestbook application uses **Redis** to store its data. It writes its data to a Redis **master** instance and reads data from multiple Redis **slave** instances.

Although the Redis master is a single pod, you can make it highly available to meet traffic demands by adding replica Redis slaves.

The guestbook application has a **web frontend** serving the HTTP requests written in PHP. It is configured to connect to the redis-master `Service` for write requests and the redis-slave service for Read requests.

First, let's create a namespace to keep our work isolated:

```
[1] kubectl create ns kui-notebook-3
```

Show Sample Output

Learning the Application's Structure

This application consists of the following components:

- A single-instance Redis master to store guestbook entries
- Multiple replicated Redis instances to serve reads
- Multiple web frontend instances

By executing `kubectl get -f <dir>`, where `<dir>` is the directory that contains the application's resource definitions, you can view a visual summary of this application structure. This summary lets you **preview the deployment**.

Here, we make sure to indicate that we want to target this application to the namespace we created on the left.

```
[1] kubectl get -f plugins/plugin-kubectl/tests/data/k8s/application/guestbook/ -n kui-notebook-3
```

Show Sample Output

Deploying the Application

By executing the following `apply -f` command view, you can initiate the deployment of this application to your cluster.

```
[2] kubectl apply -n kui-notebook-3 -f plugins/plugin-kubectl/tests/data/k8s/application/guestbook/
```

Show Sample Output

Inspecting the Running Application

By drilling down the resource in the above view, you can inspect the status of the running deployment. For example, the following sidecar is the result of clicking `Deployment` `frontend`.

```
[3] kubectl get Deployment frontend -o yaml -n kui-notebook-3
```

Now Playing: Kubernetes – Deploying Applications | PatternFly4 Light

Kui

Kui | Tab 1 | Kubernetes – Deploying Applications | Kubernetes – Working with Jobs

Kubernetes Jobs

Kubernetes supports parallel batch job scheduling via the **Job** resource type. A Job creates one or more Pods and ensures that a specified number of them successfully terminate.

In this notebook, you will learn how to create a job to run multiple tasks in parallel and use Kui's **Sequence Diagram** to inspect jobs and pods.

Controlling Parallelism

Kubernetes Jobs use two parameters, `completions` and `parallelism`, to control how a Job is subdivided into units of work. Each unit of work is executed by a **Pod**. In the example on the right, we specified values of `20` and `4`, respectively. These parameters tell Kubernetes to execute the Job across 20 Pods, but schedule at most 4 Pods to run concurrently.

You can see these values defined by expanding the `job.yaml` source.

You can also see the execution pattern visualized in the **Sequence Diagram** view. Note how there are steps of four Pods executed at a time, with a new Pod being scheduled only when one of the prior Pods has completed its work.

The gray hashed prefix of every bar represents the **cold start time** of the Pod. This is the time spent bringing up the Pod, and is in contrast to the work you actually want to accomplish. If the cold start penalty is high, you may want to consider coarsening your work: try to do more work per Pod.

Scheduling a Job

First, let's create a namespace to keep our work isolated:

```
[1] kubectl create ns kui-notebook-2
```

Show Sample Output

To schedule a job, use `kubectl apply` with a definition of your job. The following command applies a `job.yaml` to our scratch namespace. The response from this command is a live Sequence Diagram visualization.

```
[2] k apply -f plugins/plugin-kubectl/tests/data/k8s/job.yaml -n kui-notebook-2
```

Show job.yaml

Show Sample Output

This visualization shows how the sub-tasks of our Job are scheduled, over time. You can see concurrency and cold start penalties. The live table will update as tasks are scheduled and complete.

You may also switch to a list or grid view by clicking the toolbar buttons at the bottom of the view.

Inspecting the details of Job execution

To view the details of the tasks, you may click on any of the rows of the Sequence Diagram (or rows of the Table view; or cells of the Grid view). AfterKubernetes runs the tasks in Pods, so, after clicking on a task in the above view, the drilldown view you will see is a Pod view!

```
[3] kubectl get Pod pi-j272m -o yaml -n kui-notebook-2
```

Show Sample Output

Now Playing: Kubernetes – Working with Jobs | PatternFly4 Light

Kui

Kui | Tab 1 | Popeye Dashboard

Popeye Dashboard

Popeye will scan the complete cluster and report back with issues related to configurations and resources. It helps you execute the best practices on the Kubernetes cluster to stay away from common issues.

Using Popeye, you can easily identify dead and unused resources, port mismatches, RBAC rules, metrics utilization, and much more.

This notebook demonstrates three Popeye checks in the terminal to the top-right.

1. Check all resources in your cluster.
2. Check Pods in your cluster.
3. Check Deployments in your cluster.

The bottom-right terminal shows the result of clicking on one of those tables.

This notebook shows pre-recorded output, not the status of your cluster. If you have Popeye installed, you may refresh the individual commands by clicking on the green play button.

```
[1] k popeye
```

Hide Sample Output

Status Grid

Example of drilling down the Pods report in the upper middle terminal:

```
[1] kubectl popeye -s deployment default/redis-slave -n default
```

Hide Sample Output

Popeye > default > redis-slave > 2 rows

Group	STATUS	Message
slave	Warning	[POP-107] No resource limits defined
slave	OK	[POP-108] Unnamed port 6379

Now Playing: Popeye Dashboard | PatternFly4 Light