Skaffold

Balint Pato, Dan Lorenc, David Gageot, Matt Rickard, Nick Kubala, Vic Iglesias

Table of Contents

version: v0.18.0
updated: 2018-11-08
commit: 34651689be78b2c6bcfbace5072b00b93661f895

Introduction

SKAFFOLD

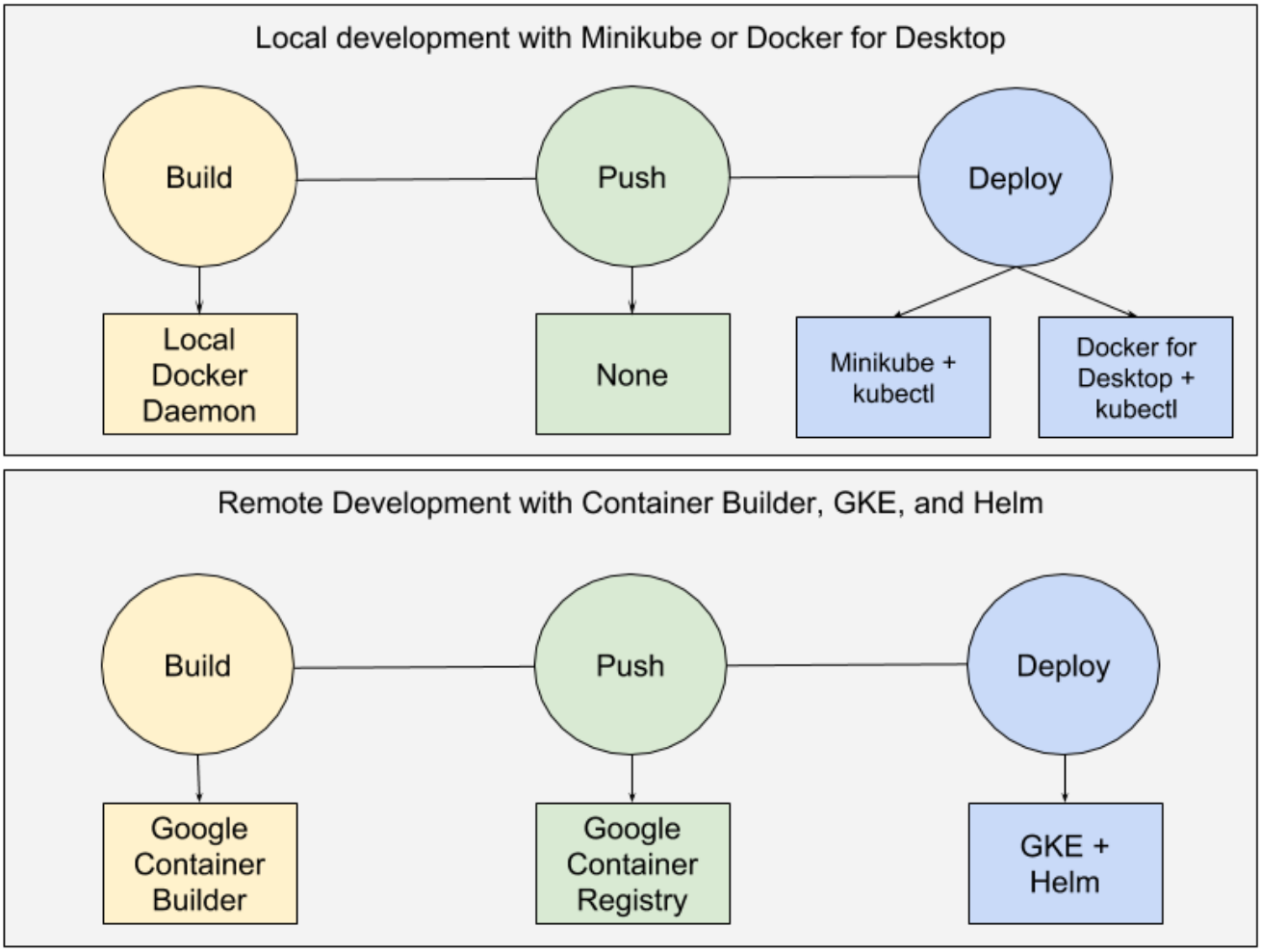
Skaffold is a command line tool that facilitates continuous development for Kubernetes applications. You can iterate on your application source code locally then deploy to local or remote Kubernetes clusters. Skaffold handles the workflow for building, pushing and deploying your application. It can also be used in an automated context such as a CI/CD pipeline to leverage the same workflow and tooling when moving applications to production.

Features

- No server-side component. No overhead to your cluster.
- Detect changes in your source code and automatically build/push/deploy.
- Image tag management. Stop worrying about updating the image tags in Kubernetes manifests to push out changes during development.
- Supports existing tooling and workflows. Build and deploy APIs make each implementation composable to support many different workflows.
- Support for multiple application components. Build and deploy only the pieces of your stack that have changed.
- Deploy regularly when saving files or run one off deployments using the same configuration.

Pluggability

Skaffold has a pluggable architecture that allows you to choose the tools in the developer workflow that work best for you.


Local development with Minikube or Docker for Desktop
Build
Push
Deploy
Local Docker Daemon
None
Minikube + kubectl
Docker for Desktop + kubectl
Remote Development with Container Builder, GKE, and Helm
Build
Push
Deploy
Google Container Builder
Google Container Registry
GKE + Helm

Operating modes

skaffold dev

Updates your deployed application continually:

- Watches your source code and the dependencies of your docker images for changes and runs a build and deploy when changes are detected
- Streams logs from deployed containers
- Continuous build-deploy loop, only warn on errors

skaffold run

Runs a Skaffold pipeline once, exits on any errors in the pipeline. Use for:

- Continuous integration or continuous deployment pipelines
- Sanity checking after iterating on your application

Getting Started with Local Tooling

For getting started with Google Kubernetes Engine and Container Builder go here. Otherwise continue below to get started with a local Kubernetes cluster.

Installation

You will need the following components to get started with Skaffold:

1. skaffold
   - To download the latest Linux build, run:
   - `curl -Lo skaffold https://storage.googleapis.com/skaffold/releases/latest/skaffold-linux-amd64 && chmod +x skaffold && sudo mv skaffold /usr/local/bin`
   - To download the latest OSX build, run:
   - `curl -Lo skaffold https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-amd64 && chmod +x skaffold && sudo mv skaffold /usr/local/bin`
2. Kubernetes Cluster
   - Minikube, GKE, Docker for Mac (Edge) and Docker for Windows (Edge) have been tested but any Kubernetes cluster will work.
3. kubectl
   - If you're not using Minikube, configure the current-context with your target cluster for development
4. docker
5. Docker image registry
   - Your docker client should be configured to push to an external docker image repository. If you're using a minikube or Docker for Desktop cluster, you can skip this requirement.
   - If you are using Google Container Registry (GCR), you can use `gcloud` and run `gcloud auth configure-docker` to configure Docker with GCR credentials. If you don't have `gcloud`, you can install a standalone helper – `docker-credential-gcr`.

Iterative Development

1. Clone this repository to get access to the examples.

```
git clone https://github.com/GoogleContainerTools/skaffold
```

2. Change directories to the `getting-started` example.

```
cd examples/getting-started
```

3. Skaffold needs a repo to push to (unless you are running against a local k8s cluster)

4. Run `skaffold dev --default-repo <your-image-repo>`.

```
$ skaffold dev --default-repo <your-image-repo>
Starting build...
Found [minikube] context, using local docker daemon.
Sending build context to Docker daemon  6.144kB
Step 1/5 : FROM golang:1.9.4-alpine3.7
 ---> fb6e10bf973b
Step 2/5 : WORKDIR
/go/src/github.com/GoogleContainerTools/skaffold/examples/getting-started
 ---> Using cache
 ---> e9d19a54595b
Step 3/5 : CMD ./app
 ---> Using cache
 ---> 154b6512c4d9
Step 4/5 : COPY main.go .
 ---> Using cache
 ---> e097086e73a7
Step 5/5 : RUN go build -o app main.go
 ---> Using cache
 ---> 9c4622e8f0e7
Successfully built 9c4622e8f0e7
Successfully tagged 930080f0965230e824a79b9e7eccffbd:latest
Successfully tagged gcr.io/k8s-skaffold/skaffold-
example:9c4622e8f0e7b5549a61a503bf73366a9cf7f7512aa8e9d64f3327a3c7fded1b
Build complete in 657.426821ms
Starting deploy...
Deploying k8s-pod.yaml...
Deploy complete in 173.770268ms
[getting-started] Hello world!
```

5. Skaffold has done the following for you:
    - Build an image from the local source code
    - Tag it with its sha256
    - Sets that image in the Kubernetes manifests defined in `skaffold.yaml`
    - Deploy the Kubernetes manifests using `kubectl apply -f`
6. You will see the output of the pod that was deployed:

```
[getting-started] Hello world!
[getting-started] Hello world!
[getting-started] Hello world!
```

Now, update `main.go`

```
diff --git a/examples/getting-started/main.go b/examples/getting-started/main.go
index 64b7bdfc..f95e053d 100644
--- a/examples/getting-started/main.go
+++ b/examples/getting-started/main.go
@@ -7,7 +7,7 @@ import (

 func main() {
        for {
-               fmt.Println("Hello world!")
+               fmt.Println("Hello jerry!")
                time.Sleep(time.Second * 1)
        }
 }
```

Once you save the file, you should see the pipeline kick off again to redeploy your application:

```
[getting-started] Hello jerry!
[getting-started] Hello jerry!
```

Run a deployment pipeline once

There may be some cases where you don't want to run build and deploy continuously. To run once, use:

```
$ skaffold run
```

Build from source TL;DR

1. Install Go (minimum required version 1.10)
2. Get the source

```
$ go get -u -d github.com/GoogleContainerTools/skaffold
```

3. Build and install Skaffold to `$GOPATH/bin/skaffold`

```
make install
```

```
skaffold vendors all of its dependencies so make install should work out of the box
```

Quick start for GKE

Prerequisites

1. GCP Account: Sign up for a free trial here.

Setup

1. Create a Kubernetes Engine cluster if you don't already have one.

```
gcloud container clusters create skaffold --zone us-west1-a
```

2. Clone the Skaffold repository then change directories to the sample application.

```
git clone https://github.com/GoogleContainerTools/skaffold.git
cd skaffold/examples/getting-started
```

3. Install `skaffold`.

```
curl -Lo skaffold https://storage.googleapis.com/skaffold/releases/latest/skaffold-
linux-amd64
chmod +x skaffold
sudo mv skaffold /usr/local/bin
```

Continuous development

The sample application you will use is a simple Go process that logs a statement every second.

As a new developer on-boarding you need to start Skaffold in `dev` mode to begin iterating on the application and seeing the updates happen in real time. The development team working on the application has already setup the Dockerfile, Kubernetes manifests, and Skaffold manifest necessary to get you started.

1. Change the references in `skaffold.yaml` and `k8s-pod.yaml` to point to your Container Registry.

```
sed -i -e s#k8s-skaffold#${GOOGLE_CLOUD_PROJECT}#g skaffold.yaml
sed -i -e s#k8s-skaffold#${GOOGLE_CLOUD_PROJECT}#g k8s-pod.yaml
```

2. Take a look at the contents of `skaffold.yaml`. You'll notice a profile named `gcb` that will be using

Google Cloud Build to build and push your image. The deploy section is configured to use kubectl to apply the Kubernetes manifests.

```
cat skaffold.yaml
```

3. Run Skaffold in `dev` mode with the `gcb` profile enabled. This will use Container Builder to build a new image from the local source code, push it to your Container Registry and then deploy your application to your Kubernetes Engine cluster.

```
skaffold dev -p gcb
```

4. You will see the application's logs printing to the screen.

```
Starting deploy...
Deploying k8s-pod.yaml...
Deploy complete.
[getting-started getting-started] Hello world!
[getting-started getting-started] Hello world!
[getting-started getting-started] Hello world!
```

5. Click the editor toggle button  in the top right of the Cloud Shell interface. The Cloud Shell editor is now open and displaying the contents of your Cloud Shell home directory.
6. Navigate to the `skaffold/examples/getting-started` directory in the left hand file navigation pane.
7. Click the `main.go` file to open it.
8. Edit the `Hello World` message to say something different. Your change will be saved automatically by the editor. Once the save is complete Skaffold will detect that a file has been changed and then rebuild, repush and redeploy the change. You will see your new log line now streaming back from the Kubernetes cluster.

Concepts

This document explains the concepts you will encounter when using Skaffold.

Skaffold configuration file or `skaffold.yaml`

The Skaffold configuration file is where the workflow is configured. In this file you define the tools you will be using and their configuration.

Phases

There are 3 main phases in the Skaffold workflow definition.

1. Build

In the build phase, Skaffold will use the tool of your choice to build an artifact. Artifacts are Docker images that you would like to deploy into your Kubernetes cluster when changes are made to them.

2. Push

In the push phase, Skaffold ensures that your image is uploaded to the registry referenced in the image name. Before running Skaffold you should ensure that you are able to push to your configured image registry.

Skaffold by default will try push the images to the image repository defined by the artifact image names. When running someone else's project, you can reroute the images to be pushed to your own image repository instead in one of the four ways:

- flag: `skaffold dev --default-repo <myrepo>`
- env var: `SKAFFOLD_DEFAULT_REPO=<myrepo> skaffold dev`
- global skaffold config (one time): `skaffold config set --global default-repo <myrepo>`
- skaffold config for current kubectl context: `skaffold config set default-repo <myrepo>`

When using local Kubernetes clusters like Minikube or Docker for Desktop, the push is skipped because the image is already available locally.

3. Deploy

The deploy step ensures that the most recent artifacts are running in the cluster. You can use different tools for deployment, for example `kubectl` or `helm`. Each deployment type has parameters that allow you to define how you want your app to be installed and updated.

Note: kubectl version 1.12.0 or greater is recommended for use with skaffold.

Artifact

Artifacts are the outputs of the build phase. Artifacts are created by running a set of steps on some source code. Currently the only artifact type is a Docker image. You can define multiple artifacts that Skaffold needs to build. When running in `dev` mode, Skaffold will only rebuild artifacts whose source code has changed. Artifacts are configured by pointing Skaffold at a Dockerfile to build and giving the image a name.

Tag Policy

Tag policies are configured in the build phase and tell Skaffold how your images should be tagged when they are pushed.

During development, it is best to have Skaffold use content based tagging strategy, `sha256`, so that changes to your source code cause Kubernetes to redeploy your new images.

When running Skaffold as part of a CI/CD pipeline you can use the `gitCommit` strategy so that your deployed resources reference the source code commit that they were deployed from.

Published Artifacts in CI

We publish a number of artifacts on each commit to the master branch

Latest Artifacts (HEAD)

Artifacts for the latest commit merged to master can always be found at

https://storage.googleapis.com/skaffold/builds/latest/skaffold-darwin-amd64
https://storage.googleapis.com/skaffold/builds/latest/skaffold-darwin-amd64.sha256
https://storage.googleapis.com/skaffold/builds/latest/skaffold-linux-amd64
https://storage.googleapis.com/skaffold/builds/latest/skaffold-linux-amd64.sha256
https://storage.googleapis.com/skaffold/builds/latest/skaffold-windows-amd64.exe
https://storage.googleapis.com/skaffold/builds/latest/skaffold-windows-amd64.exe.sha256

Documentation

https://storage.googleapis.com/skaffold/builds/latest/docs/index.html https://storage.googleapis.com/skaffold/builds/latest/docs/index.pdf

Docker image

```
gcr.io/k8s-skaffold/skaffold:latest
```

Per Commit

A skaffold docker image with most builder and deploy dependencies is tagged with each commit

```
gcr.io/k8s-skaffold/skaffold:<commit_sha>
```

Binary artifacts for a specific commit can be found at

```
https://storage.googleapis.com/skaffold/builds/<commit_sha>/skaffold-darwin-amd64
https://storage.googleapis.com/skaffold/builds/<commit_sha>/skaffold-darwin-
amd64.sha256
https://storage.googleapis.com/skaffold/builds/<commit_sha>/skaffold-linux-amd64
https://storage.googleapis.com/skaffold/builds/<commit_sha>/skaffold-linux-
amd64.sha256
https://storage.googleapis.com/skaffold/builds/<commit_sha>/skaffold-windows-amd64.exe
https://storage.googleapis.com/skaffold/builds/<commit_sha>/skaffold-windows-
amd64.exe.sha256


https://storage.googleapis.com/skaffold/builds/<commit_sha>/docs/index.html
https://storage.googleapis.com/skaffold/builds/<commit_sha>/docs/index.pdf
```

Tagged Releases

Artifacts from releases that are tagged on GitHub are also available at

```
https://storage.googleapis.com/skaffold/releases/<tag_name>/skaffold-darwin-amd64
https://storage.googleapis.com/skaffold/releases/<tag_name>/skaffold-darwin-
amd64.sha256
https://storage.googleapis.com/skaffold/releases/<tag_name>/skaffold-linux-amd64
https://storage.googleapis.com/skaffold/releases/<tag_name>/skaffold-linux-
amd64.sha256
https://storage.googleapis.com/skaffold/releases/<tag_name>/skaffold-windows-amd64.exe
https://storage.googleapis.com/skaffold/releases/<tag_name>/skaffold-windows-
amd64.exe.sha256
```

Docker image

```
gcr.io/k8s-skaffold/skaffold:<tag_name>
```

Latest Tagged Release

Artifacts from the latest release tagged on GitHub is available at

https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-amd64
https://storage.googleapis.com/skaffold/releases/latest/skaffold-darwin-amd64.sha256
https://storage.googleapis.com/skaffold/releases/latest/skaffold-linux-amd64
https://storage.googleapis.com/skaffold/releases/latest/skaffold-linux-amd64.sha256
https://storage.googleapis.com/skaffold/releases/latest/skaffold-windows-amd64.exe
https://storage.googleapis.com/skaffold/releases/latest/skaffold-windows-amd64.exe.sha256

Documentation

https://storage.googleapis.com/skaffold/releases/latest/docs/index.html
https://storage.googleapis.com/skaffold/releases/latest/docs/index.pdf

Examples

To run the examples, you either have to manually replace the image repositories in the examples from gcr.io/k8s-skaffold to yours or you can point skaffold to your default image repository in one of the four ways:

- flag: `skaffold dev --default-repo <myrepo>`
- env var: `SKAFFOLD_DEFAULT_REPO=<myrepo> skaffold dev`
- global skaffold config (one time): `skaffold config set --global default-repo <myrepo>`
- skaffold config for current kubectl context: `skaffold config set default-repo <myrepo>`

These examples are made to work with the latest release of skaffold.

If you are running skaffold at HEAD or have built it from source, please use the examples at integration/examples.

If you wish to make changes to these examples, please edit the ones at integration/examples, as those will be synced on release.

see on Github

Example: Getting started with a simple go app

This is a simple example based on

- **building** a single go file app and with a multistage `Dockerfile` using local docker to build
- **tagging** using the default tagPolicy (`gitCommit`)
- **deploying** a single container pod using `kubectl`

Example files

skaffold.yaml

```
apiVersion: skaffold/v1alpha5
kind: Config
build:
  artifacts:
  - image: gcr.io/k8s-skaffold/skaffold-example
deploy:
  kubectl:
    manifests:
      - k8s-*
```

main.go

```
package main

import (
 "fmt"
 "time"
)

func main() {
 for {
     fmt.Println("Hello world!")

     time.Sleep(time.Second * 1)
 }
}
```

Dockerfile

```
FROM golang:1.10.1-alpine3.7 as builder
COPY main.go .
RUN go build -o /app main.go

FROM alpine:3.7
CMD ["./app"]
COPY --from=builder /app .
```

k8s-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: getting-started
spec:
  containers:
  - name: getting-started
    image: gcr.io/k8s-skaffold/skaffold-example
```

Example: µSvcs with Skaffold

see on Github

In this example:

- Deploy multiple applications with skaffold
- In development, only rebuild and redeploy the artifacts that have changed
- Deploy multiple applications outside the working directory

In the real world, Kubernetes deployments will consist of multiple applications that work together. In this example, we'll walk through using skaffold to develop and deploy two applications, an exposed "web" frontend which calls an unexposed "app" backend.

WARNING: If you're running this on a cloud cluster, this example will create a service and expose a webserver. It's highly suggested that you only run this example on a local, private cluster like minikube or Kubernetes in Docker for Desktop.

Running the example on minikube

From this directory, run

```
$ skaffold dev
```

Now, in a different terminal, hit the `leeroy-web` endpoint

```
$ curl $(minikube service leeroy-web --url)
leeroooooy app!
```

Now, let's change the message in `leeroy-app` without changing `leeroy-web`. Add a few exclamations points because this is exhilarating stuff.

In `leeroy-app/app.go`, change the message here

```
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "leeroooooy app!!!\n")
}
```

Once you see the log message

```
[leeroy-app-5b4dfdcbc6-6vf6r leeroy-app] 2018/03/30 06:28:47 leeroy app server ready
```

Your service will be ready to hit again with

```
$ curl $(minikube service leeroy-web --url)
leeroooooy app!!!
```

Configuration walkthrough

Let's walk through the first part of the skaffold.yaml

```
  artifacts:
  - image: gcr.io/k8s-skaffold/leeroy-web
    context: ./leeroy-web/
  - image: gcr.io/k8s-skaffold/leeroy-app
    context: ./leeroy-app/
```

We're deploying a `leeroy-web` image, which we build in the context of its subdirectory and a `leeroy-app` image built in a similar manner.

`leeroy-web` will listen for requests, and then make a simple HTTP call to `leeroy-app` using Kubernetes service discovery and return that result.

In the deploy stanza, we use the glob matching pattern to deploy all YAML and JSON files in the respective kubernetes manifest directories.

```
deploy:
  kubectl:
    manifests:
    - ./leeroy-web/kubernetes/*
    - ./leeroy-app/kubernetes/*
```

Example: using the envTemplate tag policy

This example reuses the image name and uses an environment variable `FOO` to tag the image. The way you configure it in `skaffold.yaml` is the following build stanza:

```
build:
     artifacts:
     - image: gcr.io/k8s-skaffold/skaffold-example
     tagPolicy: ①
       envTemplate: ①
         template: "{{.IMAGE_NAME}}:{{.FOO}}" ②
```

① define tagPolicy to be envTemplate

② use go templates syntax

The `IMAGE_NAME` variable is built-in and reuses the value defined in the artifacts' `image`.

Example files

skaffold.yaml

```
apiVersion: skaffold/v1alpha5
kind: Config
build:
  artifacts:
  - image: gcr.io/k8s-skaffold/skaffold-example
  tagPolicy:
    envTemplate:
      template: "{{.IMAGE_NAME}}:{{.FOO}}"
deploy:
  kubectl:
    manifests:
      - k8s-*
```

main.go

```
package main

import (
 "fmt"
 "time"
)

func main() {
 for {
     fmt.Println("Hello world!")
     time.Sleep(time.Second * 1)
 }
}
```

Dockerfile

```
FROM golang:1.10.1-alpine3.7

CMD ["./app"]
COPY main.go .
RUN go build -o app main.go
```

k8s-pod.yaml

```
  apiVersion: v1
  kind: Pod
  metadata:
    name: getting-started
  spec:
    containers:
    - name: getting-started
      image: gcr.io/k8s-skaffold/skaffold-example
      imagePullPolicy: IfNotPresent
```

Example: helm

see on Github

Deploy multiple releases with Helm

You can deploy multiple releases with skaffold, each will need a chartPath, a values file, and namespace. Skaffold can inject intermediate build tags in the the values map in the skaffold.yaml.

Let's walk through the skaffold yaml

We'll be building an image called `skaffold-helm`, and its a dockerfile, so we'll add it to the artifacts.

```
build:
  artifacts:
  - image: skaffold-helm
```

Now, we want to deploy this image with helm. We add a new release in the helm part of the deploy stanza.

```
deploy:
  helm:
    releases:
    - name: skaffold-helm
      chartPath: skaffold-helm
      namespace: skaffold
      values:
        image: skaffold-helm
      valuesFiles:
        - helm-values-file.yaml
```

This part tells skaffold to set the `image` parameter of the values file to the built skaffold-helm image and tag.

```
    values:
      image: skaffold-helm
```

Example: kustomize

This is an example demonstrating how skaffold can work with kustomize with the `skaffold deploy` command.

Example files

see on Github

skaffold.yaml

```
apiVersion: skaffold/v1alpha5
kind: Config
deploy:
  kustomize: {}
```

kustomization.yaml

```
resources:
  - deployment.yaml
patches:
  - patch.yaml
```

patch.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kustomize-test
spec:
  template:
    spec:
      containers:
      - name: kustomize-test
        image: index.docker.io/library/busybox
        command:
          - sleep
          - "3600"
```

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kustomize-test
  labels:
    app: kustomize-test
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kustomize-test
  template:
    metadata:
      labels:
        app: kustomize-test
    spec:
      containers:
      - name: kustomize-test
        image: gcr.io/k8s-skaffold/not/a/valid/image
```

Example: kaniko

This is an example demonstrating

- **building** a single go file app and with a single stage `Dockerfile` using kaniko to build on a K8S cluster
- **tagging** using the default tagPolicy (`gitCommit`)
- **deploying** a single container pod using `kubectl`

Example files

see on Github

skaffold.yaml

```
apiVersion: skaffold/v1alpha5
kind: Config
build:
  artifacts:
  - image: gcr.io/k8s-skaffold/skaffold-example
  kaniko:
    buildContext:
      gcsBucket: skaffold-kaniko
    pullSecretName: e2esecret
    namespace: default
deploy:
  kubectl:
    manifests:
      - k8s-*
```

main.go

```
package main

import (
 "fmt"
 "time"
)

func main() {
 for {
     fmt.Println("Hello world!")
     time.Sleep(time.Second * 1)
 }
}
```

Dockerfile

```
FROM gcr.io/google-appengine/golang

WORKDIR /go/src/github.com/GoogleCloudPlatform/skaffold
CMD ["./app"]
COPY main.go .
RUN go build -o app main.go
```

k8s-pod.yaml

```
  apiVersion: v1
  kind: Pod
  metadata:
    name: getting-started-kaniko
  spec:
    containers:
    - name: getting-started
      image: gcr.io/k8s-skaffold/skaffold-example
```

Example: bazel

Bazel is one of the supported builders in Skaffold. The way you configure it in `skaffold.yaml` is the following build stanza:

```
build:
     artifacts:
     - image: gcr.io/k8s-skaffold/skaffold-example
       context: . ①
       bazel: ②
         target: //:skaffold_example.tar ③
```

① make sure the context contains the bazel files (`WORKSPACE`, `BUILD`)

② add bazel to each artifact

③ specify target - our builder will use this to load to the image to the Docker daemon

Example files

skaffold.yaml

```
  apiVersion: skaffold/v1alpha5
  kind: Config
  build:
    artifacts:
    - image: gcr.io/k8s-skaffold/skaffold-bazel
      context: .
      bazel:
        target: //:skaffold_example.tar
```

main.go

```
package main

import (
 "fmt"
 "time"
)

func main() {
 for {
     fmt.Println("Hello bazel!!!!")
     time.Sleep(time.Second * 1)
 }
}
```

WORKSPACE

```
workspace(name = "skaffold")

git_repository(
    name = "io_bazel_rules_docker",
    remote = "https://github.com/bazelbuild/rules_docker.git",
    tag = "v0.4.0",
)

http_archive(
    name = "io_bazel_rules_go",
    sha256 = "feba3278c13cde8d67e341a837f69a029f698d7a27ddbb2a202be7a10b22142a",
    url =
"https://github.com/bazelbuild/rules_go/releases/download/0.10.3/rules_go-
0.10.3.tar.gz",
)

load("@io_bazel_rules_go//go:def.bzl", "go_rules_dependencies",
"go_register_toolchains")

go_rules_dependencies()
go_register_toolchains(
    go_version = "1.10.1",
)

load(
    "@io_bazel_rules_docker//go:image.bzl",
    _go_image_repos = "repositories",
)

_go_image_repos()
```

BUILD

```
load("@io_bazel_rules_docker//go:image.bzl", "go_image")

go_image(
    name = "skaffold_example",
    srcs = ["main.go"],
    goos = "linux",
    goarch = "amd64",
    static = "on",
)
```

Contributing

See CONTRIBUTING.md

Development

See DEVELOPMENT.md

Code of conduct

See code-of-conduct.md

Resources

Changelog

See CHANGELOG.md

Community

- skaffold-users mailing list
- #skaffold on Kubernetes Slack

There is a bi-weekly Skaffold users meeting at 9:30am-10am PST hosted on hangouts under "skaffold". Everyone is welcome to add suggestions to the agenda and attend. Join the skaffold-users mailing list to get the calendar invite directly on your calendar.

Copyright