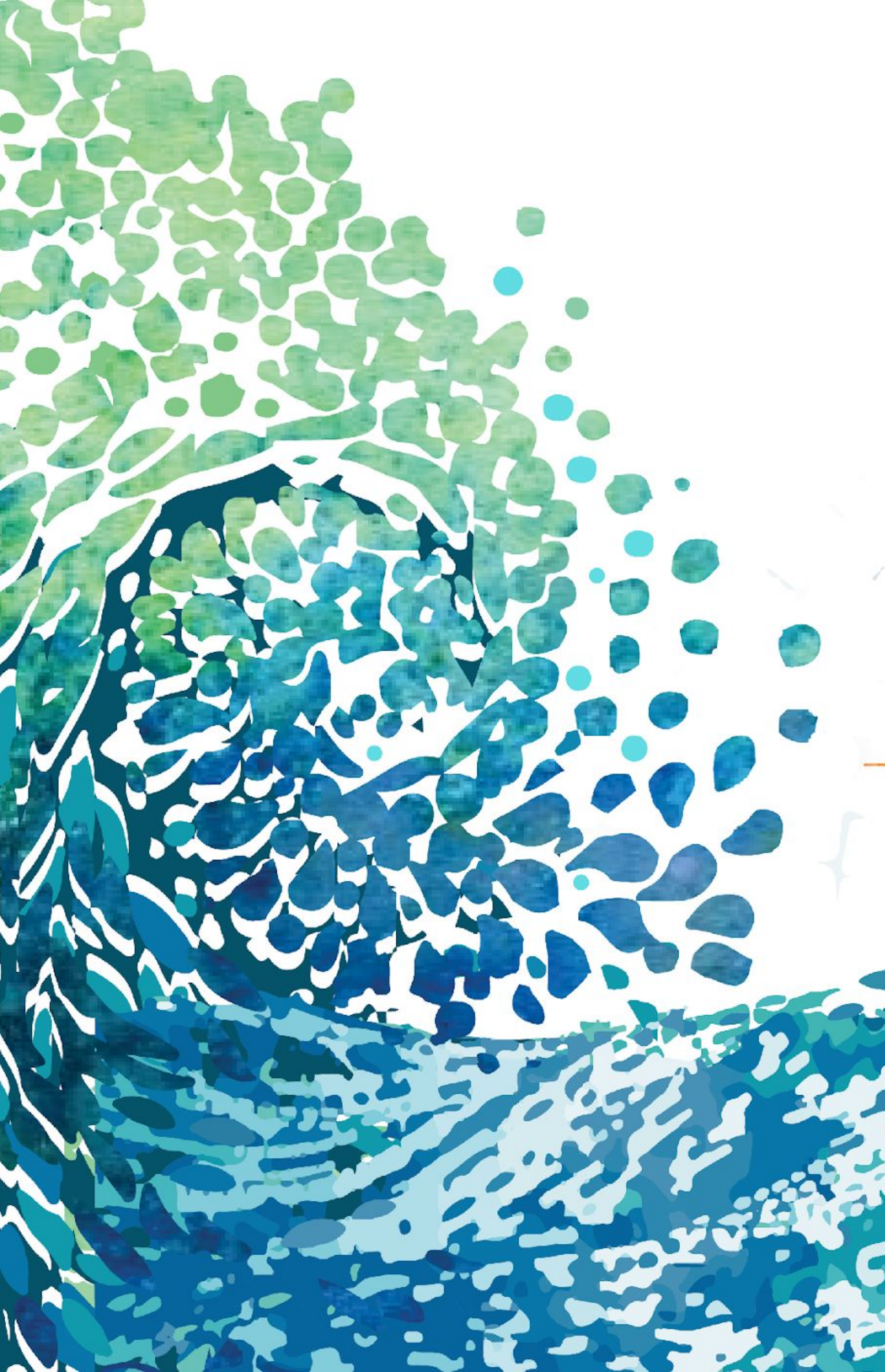
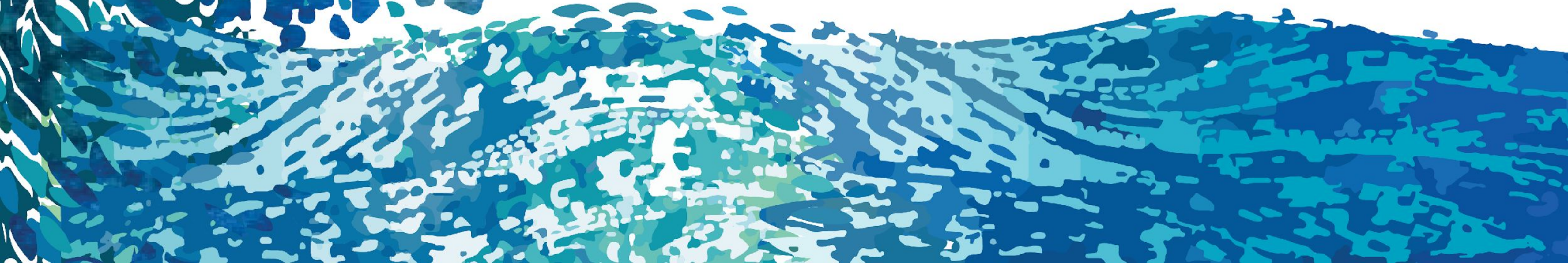




KubeCon | CloudNativeCon

Europe 2019

KubeCon | CloudNativeCon

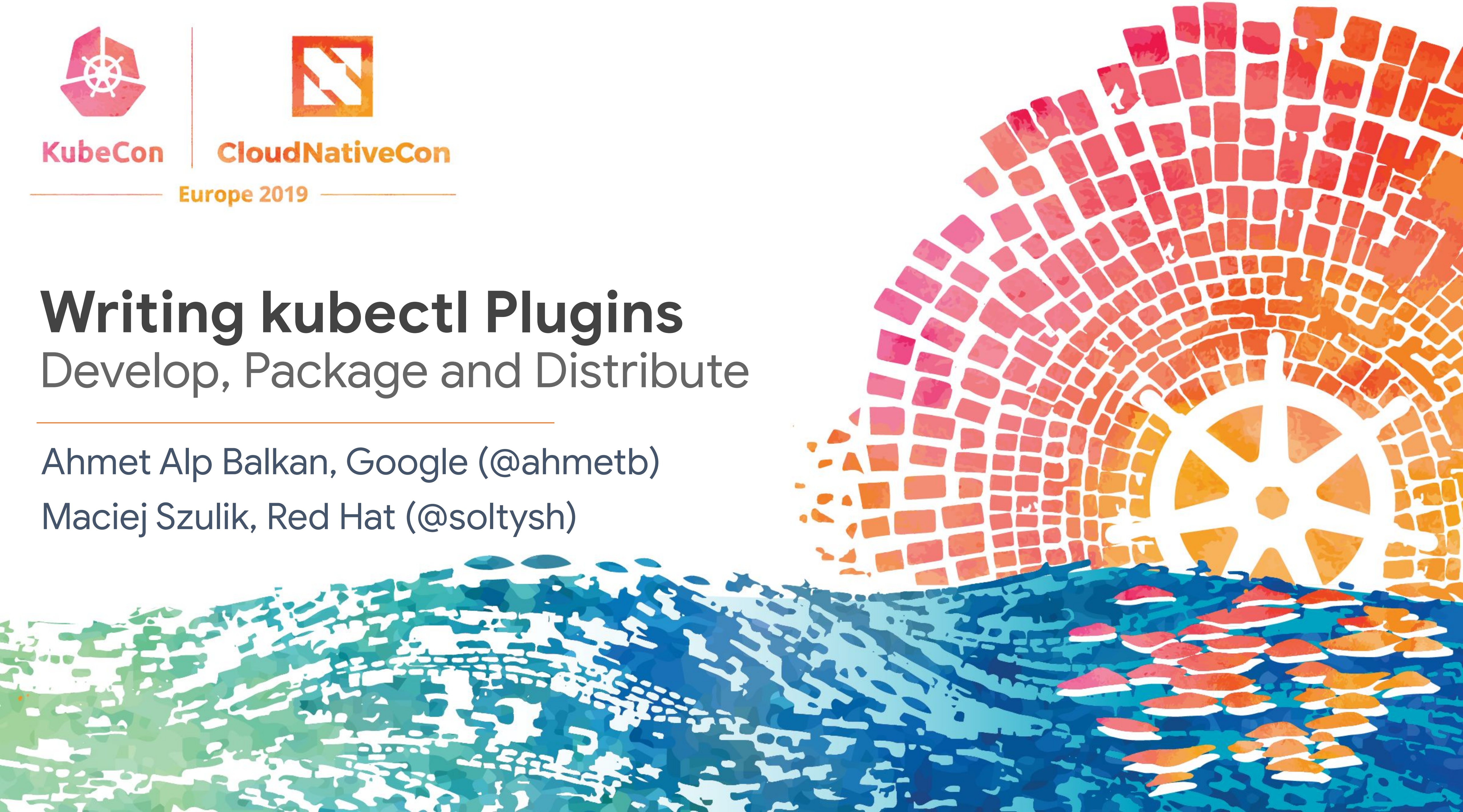
Europe 2019

Writing kubectl Plugins

Develop, Package and Distribute

Ahmet Alp Balkan, Google (@ahmetb)

Maciej Szulik, Red Hat (@soltysh)



Kubernetes

An abstraction layer for cloud infrastructure

A framework for declarative APIs and distributed control

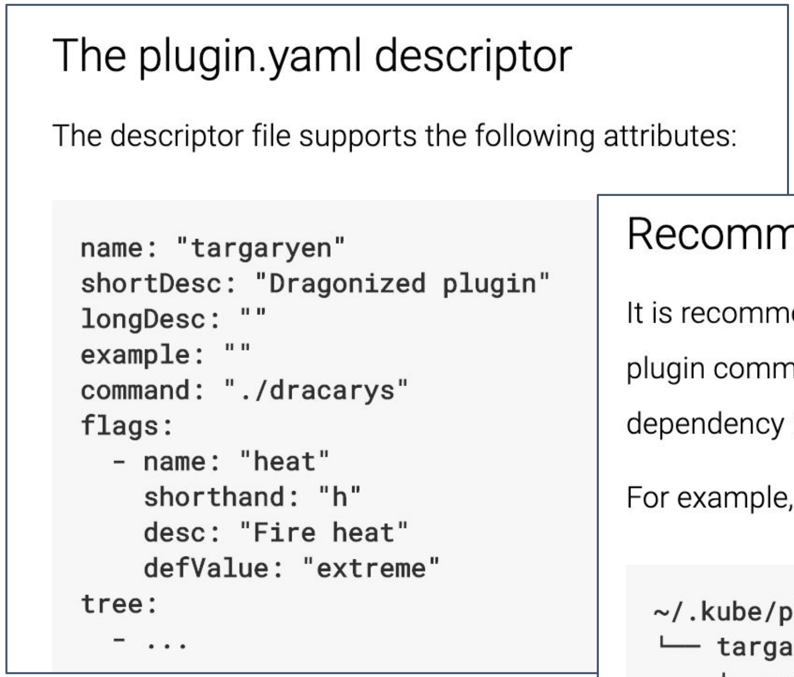
Infrastructure extensibility

API extensibility

https://kubernetes.io/docs/concepts/extend-kubernetes/

Old plugins model

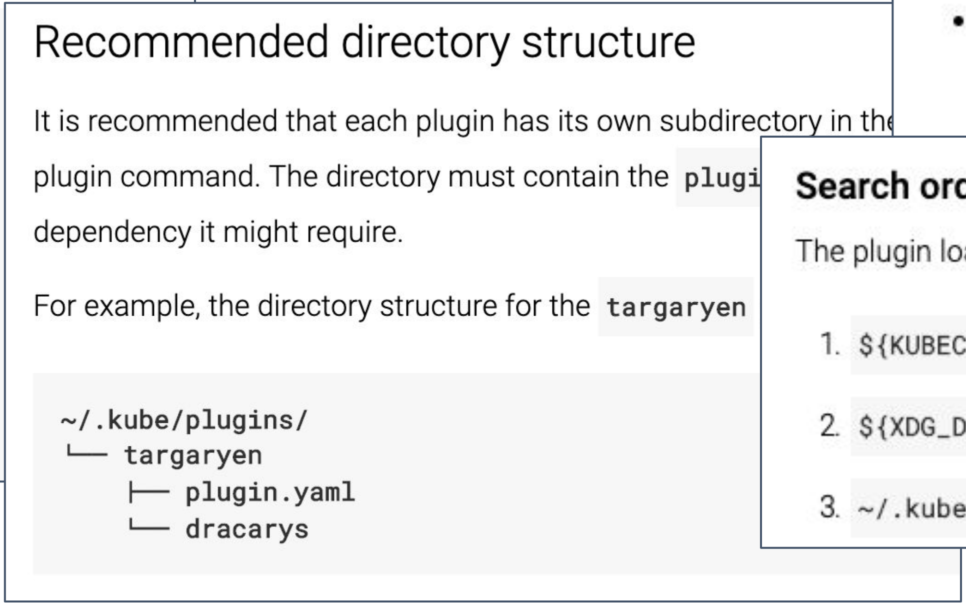
If you developed/used kubectl plugins
before kubectl 1.12 (Sep'2018) - **everything has changed.**



The plugin.yaml descriptor

The descriptor file supports the following attributes:

```
name: "targaryen"
shortDesc: "Dragonized plugin"
longDesc: ""
example: ""
command: "./dracarys"
flags:
  - name: "heat"
    shorthand: "h"
    desc: "Fire heat"
    defValue: "extreme"
tree:
  - ...
```

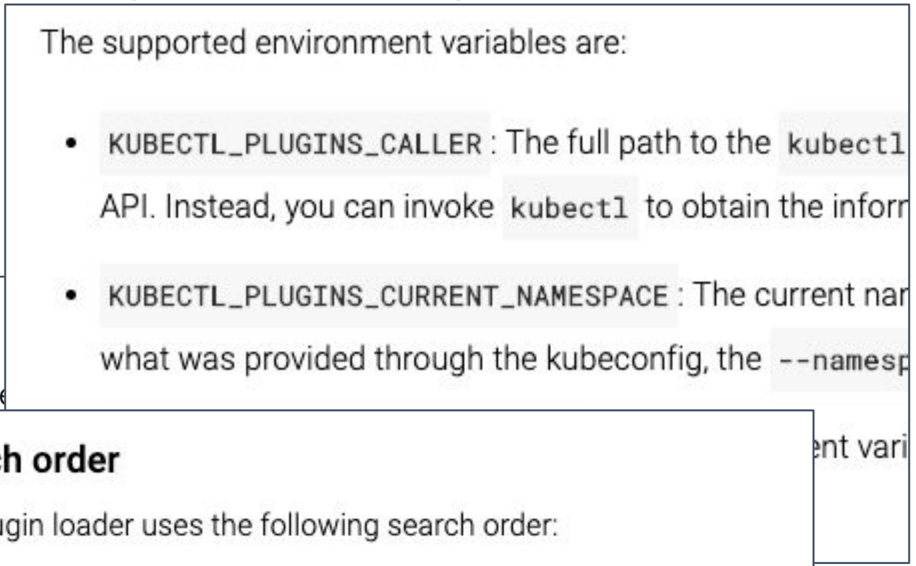


Recommended directory structure

It is recommended that each plugin has its own subdirectory in the
plugin command. The directory must contain the `plugi`
dependency it might require.

For example, the directory structure for the `targaryen`

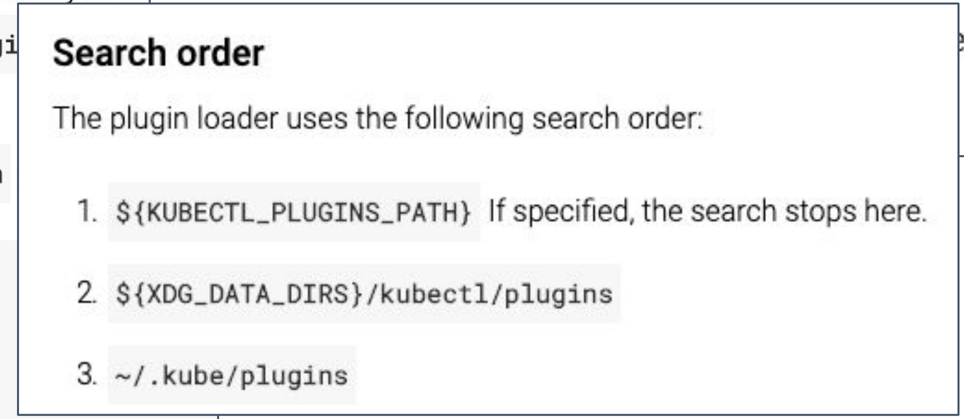
```
~/.kube/plugins/
└── targaryen
    ├── plugin.yaml
    └── dracarys
```



The supported environment variables are:

- `KUBECTL_PLUGINS_CALLER` : The full path to the `kubectl`
  API. Instead, you can invoke `kubectl` to obtain the inforr
- `KUBECTL_PLUGINS_CURRENT_NAMESPACE` : The current nar
  what was provided through the kubeconfig, the `--namesp`

ent vari



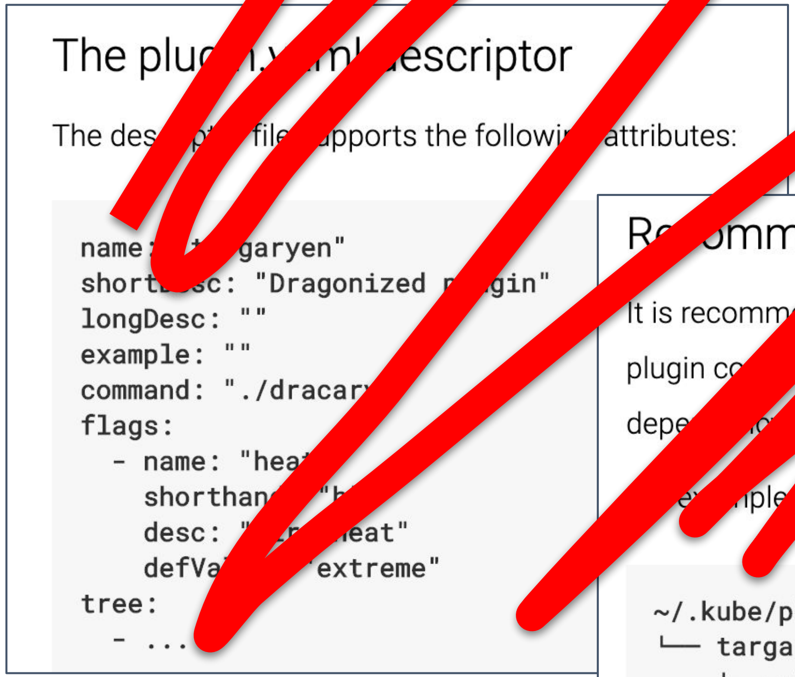
Search order

The plugin loader uses the following search order:

1. `${KUBECTL_PLUGINS_PATH}` If specified, the search stops here.
2. `${XDG_DATA_DIRS}/kubectl/plugins`
3. `~/.kube/plugins`

Old plugins model

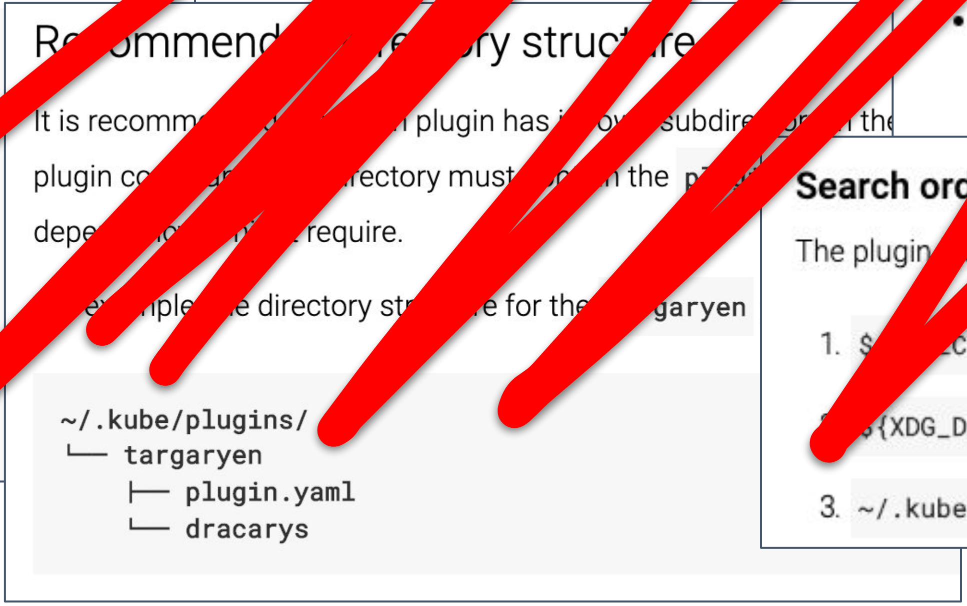
If you developed/used kubectl plugins
before kubectl 1.12 (Sep'2018), **everything has changed.**



The plugin.yaml descriptor

The descriptor file supports the following attributes:

```
name: "targaryen"
shortDesc: "Dragonized plugin"
longDesc: ""
example: ""
command: "./dracarys"
flags:
  - name: "heat"
    shorthand: "h"
    desc: "Fire heat"
    defValue: "extreme"
tree:
  - ...
```



Recommended directory structure

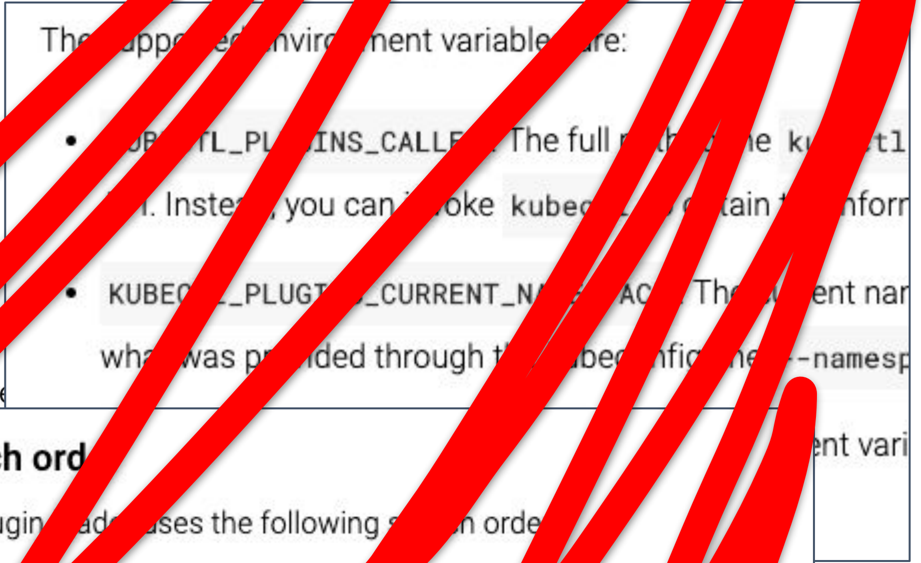
It is recommended that each plugin has its own subdirectory in the

plugin command ... directory must contain the plugin.yaml

dependencies ... require.

Example of the directory structure for the targaryen

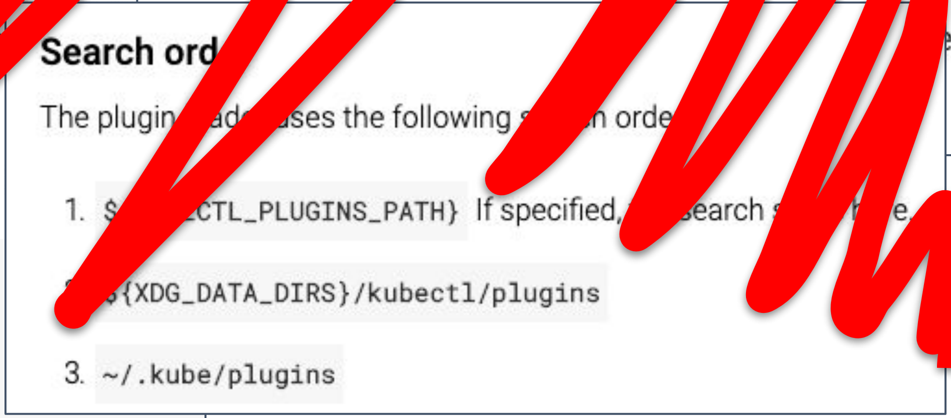
```
~/.kube/plugins/
└── targaryen
    ├── plugin.yaml
    └── dracarys
```



The supported environment variables are:

- KUBECTL_PLUGINS_CALLER: The full path to the kubectl
  . Instead, you can invoke kubectl ... to obtain the inform
- KUBECTL_PLUGINS_CURRENT_NAMESPACE: The current nam
  what was provided through the kubeconfig, the --namesp

ent vari



Search order

The plugin loader uses the following search order:

1. `${KUBECTL_PLUGINS_PATH}` If specified, the search stops here.
2. `${XDG_DATA_DIRS}/kubectl/plugins`
3. `~/.kube/plugins`

What?

An **extension mechanism** that lets you write **your own `kubectl` subcommands**

Why develop plugins?

Enhance kubectl functionality

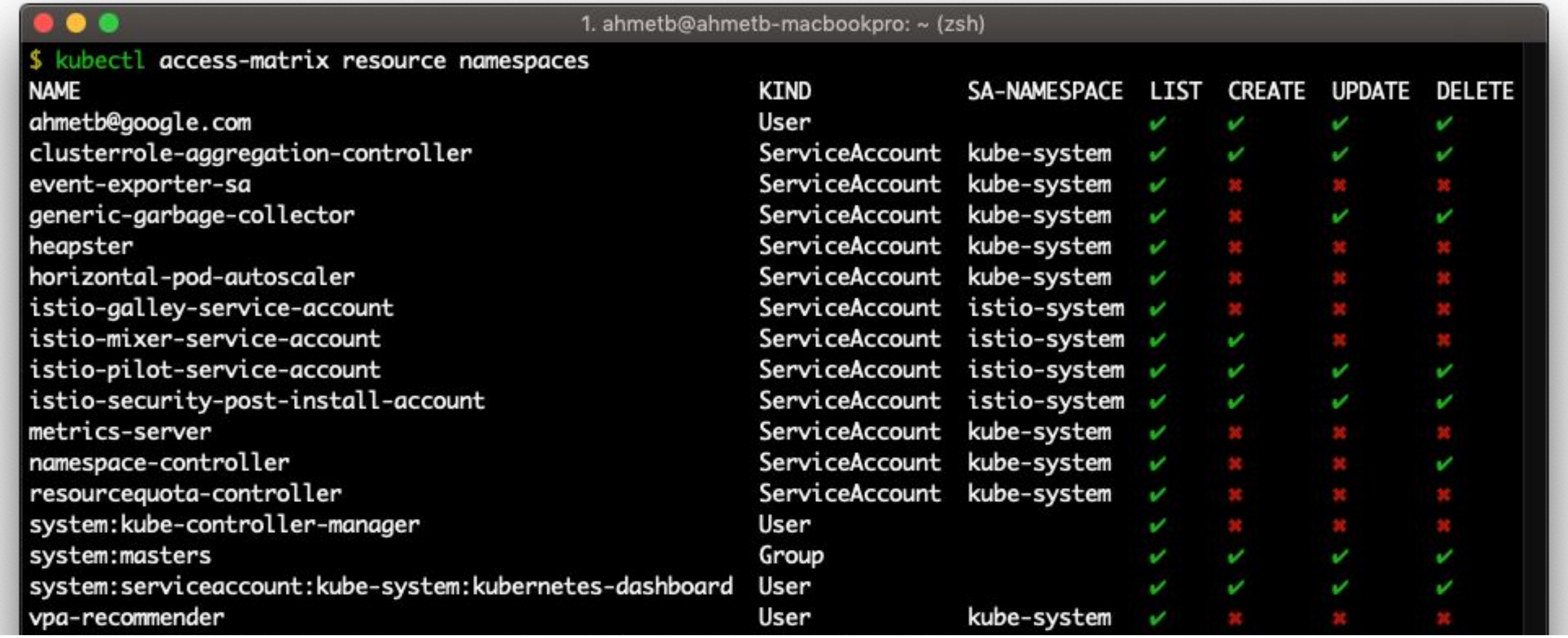
Official subcommands vs **plugins**

Feels more **natural**

Encapsulate **custom workflows**

Why #1: enhance kubectl

<u>Problem:</u> need a command to list users with RBAC permissions to an object



1. ahmetb@ahmetb-macbookpro: ~ (zsh)

```
$ kubectl access-matrix resource namespaces
NAME                                                   KIND            SA-NAMESPACE  LIST  CREATE  UPDATE  DELETE
ahmetb@google.com                                      User                          ✔     ✔       ✔       ✔
clusterrole-aggregation-controller                     ServiceAccount  kube-system   ✔     ✔       ✔       ✔
event-exporter-sa                                      ServiceAccount  kube-system   ✔     ✖       ✖       ✖
generic-garbage-collector                              ServiceAccount  kube-system   ✔     ✖       ✔       ✔
heapster                                               ServiceAccount  kube-system   ✔     ✖       ✖       ✖
horizontal-pod-autoscaler                              ServiceAccount  kube-system   ✔     ✖       ✖       ✖
istio-galley-service-account                           ServiceAccount  istio-system  ✔     ✖       ✖       ✖
istio-mixer-service-account                            ServiceAccount  istio-system  ✔     ✔       ✖       ✖
istio-pilot-service-account                            ServiceAccount  istio-system  ✔     ✔       ✔       ✔
istio-security-post-install-account                    ServiceAccount  istio-system  ✔     ✔       ✔       ✔
metrics-server                                         ServiceAccount  kube-system   ✔     ✖       ✖       ✖
namespace-controller                                   ServiceAccount  kube-system   ✔     ✖       ✖       ✔
resourcequota-controller                               ServiceAccount  kube-system   ✔     ✖       ✖       ✖
system:kube-controller-manager                         User                          ✔     ✖       ✖       ✖
system:masters                                         Group                         ✔     ✔       ✔       ✔
system:serviceaccount:kube-system:kubernetes-dashboard User                          ✔     ✔       ✔       ✔
vpa-recommender                                        User            kube-system   ✔     ✖       ✖       ✖
```

Why #2: official command vs plugin

Official command	Plugin
KEP + approval	no approvals
usefulness and stability	no restrictions
hosted in kubectl codebase (Go only)	any language
tied to Kubernetes release cycles	release at your own pace
has to be consistent with kubectl	has room for creativity
takes O(months)...O(years) from alpha→beta→stable	develop & distribute in **O(hours)**

Why #3: plugin vs standalone

rakess	→	kubectl access-matrix
kail	→	kubectl tail
ketall	→	kubectl get-all
ksort	→	kubectl sort-manifests

- ✓ Plugin names are more intuitive
- ✓ Calling via `kubectl` looks more natural
- ✓ You can discover available plugins

Why #4: encapsulate workflows

```
./install-debug-tools.sh → kubectl debug-pod
./rsync-to-pod.py        → kubectl rsync-to-pod
./force-drain-node.sh    → kubectl force-drain
```

✓ Install these on all your developers’ machines

✓ All scripts are **organized** under `kubectl` umbrella for **discoverability**

How?

Write code in **any language**

Name it `kubectl-foo`

Place in your `$PATH`

Invoke `kubectl foo`

How plugins work?

KubeCon CloudNativeCon Europe 2019

kubectl makes an execve system call

(**replaces** the kubectl process **with your plugin** executable)

Plugin process will:

- ✓ inherit the environment variables
- ✓ inherit the standard streams
- ✓ determine the exit code of the kubectl invocation

Demo: sample plugin

git.k8s.io/sample-cli-plugin

What's next?

Consistency with kubectl

Packaging and **distribution**

Updates

Consistency

Plugins should follow **kubectl** idioms and standards:

- `-n/--namespace`
- `-o/--output=[json,yaml,jsonpath,…]`
- `--kubeconfig`
- idiomatic naming for subcommands and flags
- minimal to no docs

How to be consistent?

git.k8s.io/cli-runtime: set of helpers for creating commands

- reading configuration + clients
- printing flags + utils
- polymorphic helpers

Naming

Descriptive

kubectl **sort** → kubectl **sort-manifests**

Unique

kubectl **login** → kubectl **oidc-login**

Leads with **verb+action**

kubectl **svc-open** → kubectl **open-svc**

(For more, search: Plugin Naming Style Guide)

Naming

```
kubectl-foo                  kubectl foo
kubectl-foo-bar        ⟶     kubectl foo bar
kubectl-my_plugin            kubectl my-plugin
```

(For more, see: KEP24 kubectl plugins)

Problem: plugin management

kubectl **does not** provide a solution for

...**users** to:

- install plugins
- keep them up to date
- remove plugins cleanly

...**developers** to:

- make their plugins discoverable by users
- package their plugins for multiple platforms

so we had to do something...

Meet Krew

Krew is developed at **Google** in summer of 2018 as an intern project.

Luk Burchard

Krew simplifies **plugin** usage and distribution for **users** and **developers**.

It's a SIG CLI sub-project since April'19.

sigs.k8s.io/krew

Demo: plugin user

Let’s try to use Krew as a kubectl user.

Krew overview

- **No dependency management**
- Can install only the latest version
- Has a **centralized** plugin **index**.
  - great for discoverability, slower curation, more enforcement
  - doesn't come with any security guarantees
  - soon to allow decentralized repos
- Supports Windows, macOS, Linux

Packaging with krew

1. Publicly accessible **archive file**
2. **Plugin manifest**
3. Verify manifest **locally**
4. PR to **krew-index** repository

Demo: plugin developer

Package and distribute your plugin.

Plugin manifests

```
apiVersion: krew.googlecontainertools.github.com/v1alpha2
kind: Plugin
metadata:
  name: access-matrix
spec:
  version: "v0.4.0"
  platforms:
  - ...
```

Plugin manifests

```yaml
apiVersion: krew.googlecontainertools.github.com/v1alpha2
kind: Plugin
metadata:
  name: access-matrix
spec:
  version: "v0.4.0"
  platforms:
  - selector:
      matchLabels:
        os: linux
        arch: amd64
    uri: https://github.com/corneliusweig/rakkess/releases/v0.4.0/bundle.tar.gz
    sha256: 7a16c61dfc4e2924fdedc894d59db7820bc4643a58d9a853c4eb83eadd4deee8
    files:
      - from: ./rakkess-linux-amd64
        to: "."
    bin: rakkess-linux-amd64
  - selector: ...
```

THIS IS
THE SIGN
YOU'VE BEEN
LOOKING FOR
NOSE JEWELRY

Call to action

Let’s have **more** of it

Get **creative and develop** new plugins

Rebrand your standalone tool

Help us **set the standards** for plugins

How to get involved / contact

Become a Krew contributor:

- sigs.k8s.io/krew

Join us:

- SIG CLI Meetings:
  - Biweekly on Wednesdays at 06:00 CEST/ 12:00 EDT / 09:00 PT
- SIG CLI Slack Channel:
  - #sig-cli
- SIG CLI Mailing list:
  - kubernetes-sig-cli@googlegroups.com