Lightweight, standalone and composable Che workspaces with Kubernetes Operators

David Festal
Software Engineer at Red Hat
Twitter: @dfatwork
Github: davidfestal
First, what is Che about?
“First open-source Kubernetes-native IDE…”
(Che 7 Announcement)

...for developer teams
(Che home page @ eclipse.org)
Demo

Let’s discover Che
Let’s discover Che
On OpenShift: for the ease, but would be the same on other Kubernetes envs.

Create a workspace from the Dashboard and play 2 minutes inside it to showcase the power.

Show when we really arrive *inside* the workspace (!= from the Dashboard).

Show that the workspace is only containerized Kubernetes resources + external access through services.

Switch to show the Che server => resources used!...

... though the Che server only manages:

- Workspace lifecycle
- Authentication and identity management
- Creates and manages workspace underlying K8S resources
If only we had Lightweight, Standalone Workspaces As first-class K8S citizens?
Split Che Architecture

Browser

Classical monolithic apps
- Identity and Access Management
- Data Store
- Plug-ins Registry
- Stacks Registry

Che Server
- Authentication
- Workspace Scheduler
- Scalability
- Health Check
- Monitoring
- APIs

Che Workspace
- Plug-in Brokers
- IDE
  - Monaco Editor
  - Menus
  - Command Palette
  - Panels
  - Languages
  - Debuggers
  - Tools

App Runtimes
- Language Servers
- Debug Servers
- VS Code Extensions
- Tooling Services

Container Orchestrator
- kubernetes
- OPENSHIFT

Storage
Split Che Architecture

Browser

Che Server
- Authentication
- Workspace Scheduler
- Scalability
- Health Check
- Monitoring

Che Workspace
- Plug-in Brokers
- IDE
  - Monaco Editor
  - Menus
  - Command Palette
  - Panels
  - Languages
  - Debuggers
  - Tools

THEIA

Container Orchestration
- kubernetes
- Red Hat OpenShift

Containerized K8S resources

Data Store

Plug-ins Registry

Stacks Registry

Identity and Access Management
Extend Kubernetes with workspaces?

Currently

➡️ Workspace drives the creation of standard Kubernetes objects

Go one step further

➡️ Make workspaces themselves become Kubernetes objects

Now Possible since Kubernetes has become extensible with Custom Resources, and finally Operators

EclipseCon Europe 2019
A word on Kubernetes Operators

**Custom Resource Definitions (CRD)**
- Allow new types *K8S custom resources (CR)*: Metadata, Spec, Status

**Controllers**
- Contain logic to *reconcile expected and observed state* of the CR on CRUD events

**Operators**
- Bundle *CRDs and controller* in a single unit along with metadata and permissions
What we currently have

**Workspace Data**
- Workspace structure
- Workspace runtime desired state
- Workspace runtime effective state

All stored in Che own dedicated database

**Workspace Management logic**
- From inside a single dedicated server
- In-house *management* of K8S objects
- Sync with the internal workspace runtime status
What we could have

Workspace Data

- Workspace structure
- Workspace runtime desired state
- Workspace runtime effective state

All stored in Che own dedicated database as standard K8S objects in the cluster Etcd instance

Workspace Management logic

- From inside a single dedicated server
- In-house management of K8S objects
- Sync with the internal workspace runtime status

Che server specific model
K8S Custom Resource (CR) with:
- Devfile (already K8S-compliant syntax)
- CR Spec fields
- CR Status fields

Typical CRUD K8S resource reconciliation
Runtime state is in the workspace CR
We have it in fact... in early development

Che Workspace Operator

- Simple K8S Deployment with
  - Small GO application
  - Embedded plugin registry (apache server + yaml files)

Open-source POC hosted in the che-incubator GitHub repository

- Needs cleaning, documentation, decent testing,
- But concept and main implementation skeleton are there.

Same user experience inside the Workspace
Demo

> kubectl apply -f workspace.yaml
> kubectl apply -f workspace.yaml
Demo Content

- Kubectl -apply workspace, start, stop, wait for availability, list workspaces
- Play a bit inside such a started workspace and show it provides the same experience
- Go to the operator POD In Openshift => Show resource consumption: 10 times less
Lightweight, standalone and ... composable

Workspaces CRs become **components**, building blocks

- Can be viewed, created, managed by any process running on Kubernetes.
- Inherit K8S ecosystem goodies available for any K8S resource
  - Monitoring,
  - Event aggregating,
  - OpenApi schema validation, etc ...

**Example of integration**

- The current OpenShift console web application.
Demo

Manage workspaces in the OpenShift console
Manage workspaces in the OpenShift console
No more external components?

No more dedicated Database?

- Delegated to Kubernetes infrastructure
- Everything is stored in K8S (ETCD) thanks to the Custom Resource

No more Che server: where is the API?

- All the required info is in the CR ⇒ API == read the CR
- That can be made in a tiny workspace sidecar (Quarkus App)

No more dedicated identity management? ...
Lost authentication ?...

... Delegate to Kubernetes
Delegating authentication to K8S

**Problem**

- Many different and incompatible implementations of identity management

**Solution**

- **Switch controllers** that manage a Custom Resource (cf. Ingress class)
- **Decompose** Custom resources
  - **Workspace** == containers, basic services
  - **Workspace Exposure** == how the workspace is exposed on the network
- **Provide several controllers**
  - **Main controller** for the workspace custom resource
  - **Configurable exposure controller**, according to exposure class
Demo

Switch to OpenShift integrated authentication
Switch to OpenShift integrated authentication
Demo: Use OpenShift integrated authentication

- Restart the workspace with the openshift-oauth
- Show that it now requires authentication
Why not make it even more extensible

Same design principle could be applied to other areas

Authorization
- Delegate to RBAC, ABAC, ...

Network access
- K8S Ingresses, Openshift Routes, ISTIO (to trace workspace internals ?)

Even deployment mode
- KNative for scale-to-zero workspaces ?
Finally, all that fuss for what?
Why would we continue this way?

- Because we like kubectl?
- Only to allow quick setup of personal, standalone Che workspace?
- For the fun?
- To be trendy?

**NOT ONLY**

There are important stakes behind this new way of creating Che workspaces.

And obviously benefits for the overall Che solution.
Big enabler for the future of Che overall

No external central components required, apart from Kubernetes ones

➤ Makes it easy to **integrate** and to **manage**
➤ Allows **saving resources**

Use Kubernetes design-patterns from the ground

➤ Enables **Massive Scalability**
➤ Opens the **Kubernetes ecosystem** to Che
➤ Unlocks **Devfile Extensibility** through **custom components**
Why would we continue this way?

In order to

Enable the **widest spread of Che workspaces**
Throughout **all sorts of Kubernetes infrastructures** and flavors and
Inside **all sorts of software contexts** running on Kubernetes

While

Reducing their overall **management and maintenance burden**

And

Improving **scalability and performance**

That would be worth it to make Che even more Kube-native
Questions ?
Thank You