adaptTo()

APACHE SLING & FRIENDS TECH MEETUP
BERLIN, 25-27 SEPTEMBER 2017

0DT deployments for Sling-based apps

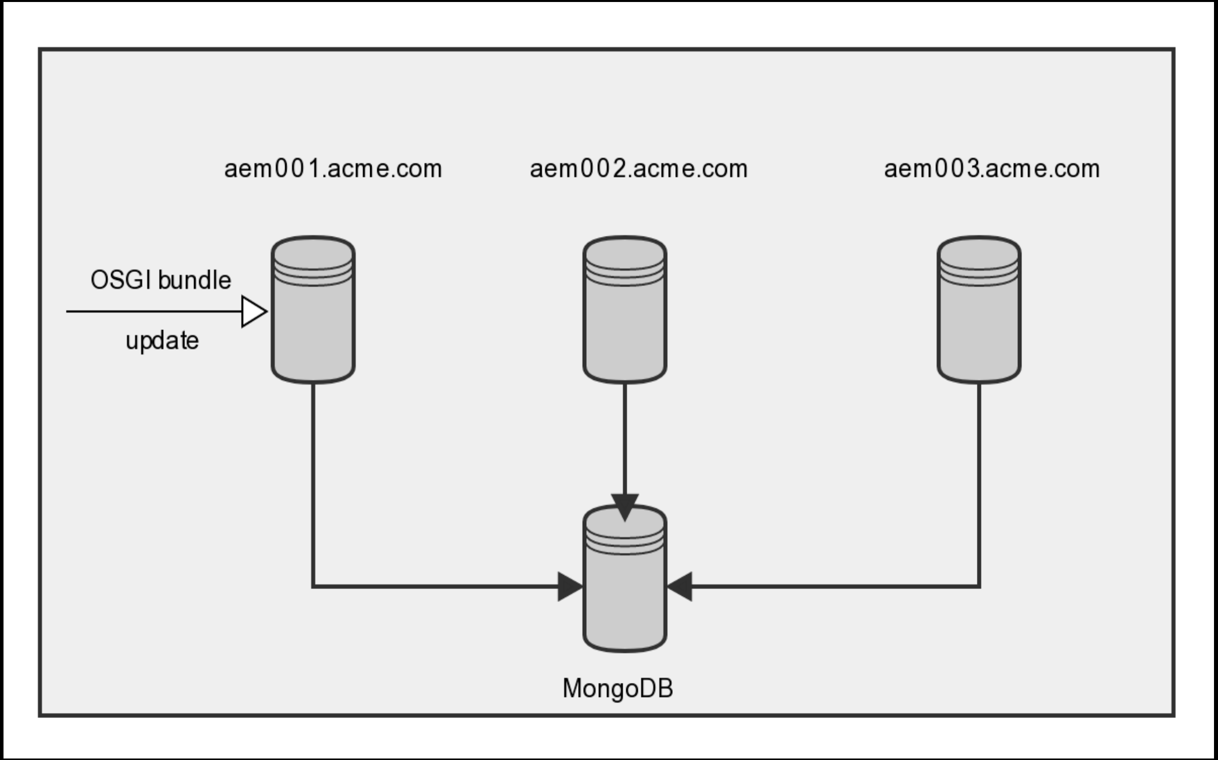
Robert Munteanu, Tomek Rękawek @ Adobe

Agenda

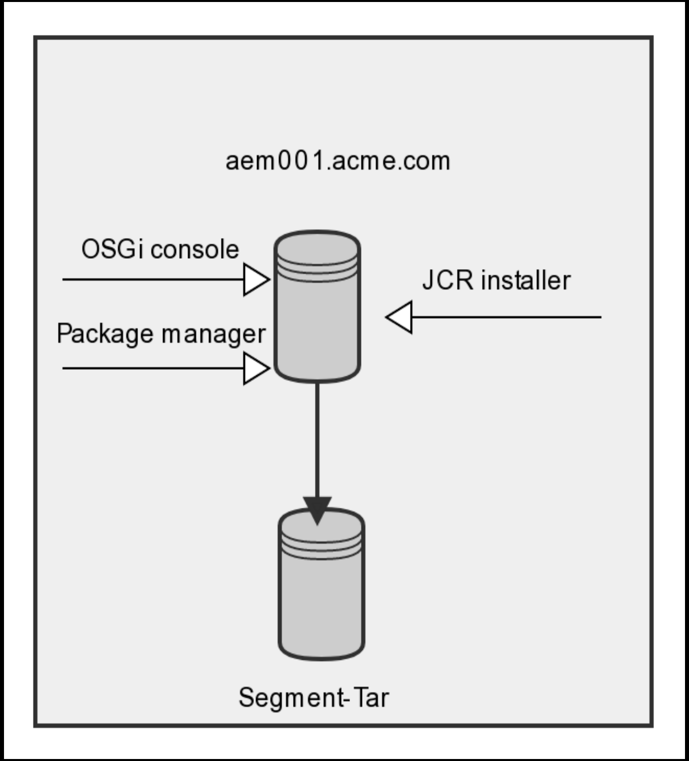
- Motivation
- Concepts
- Usage
- Demo
- Wrap-up

Motivation

Clustered deployments



Standalone deployments



adaptTo()

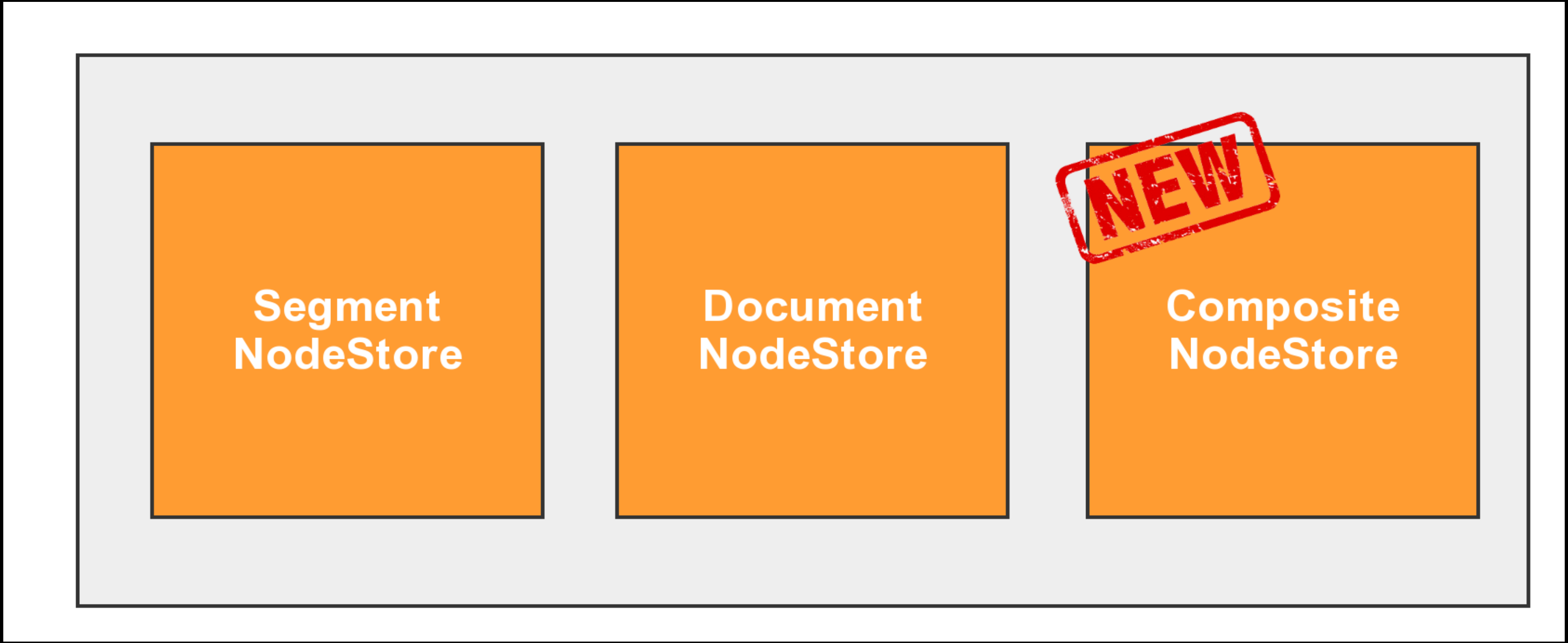
Concepts

Separate content from code

```
/
/content
/etc
/var
/tmp
/libs
/apps
```

Hold content in the main repository and put active elements in a different repository.

Oak Composite NodeStore



Oak Composite NodeStore

- Manages a number of 1 to n NodeStores
- Always has a *global* NodeStore, holding the content not claimed by other NodeStores
- Multiple mounted NodeStores, owning content for certain paths, for instance `/libs` and `/apps`
- Certain paths in the global mount can be claimed by mounts
  - Looks for a *:oak:mount-* prefix in node names and matches them with mounts
  - Currently implemented for indexes

adaptTo()

Typical configuration

Note: for your viewing pleasure, not an actual config file

```
/         <default> # global mount
/libs     libs      # read-only mount
/apps     libs      # read-only mount
```

Following paths also belong to the *libs* mount:

- `/oak:index/uuid/:oak:mount-libs-index` (and others)
- `/jcr:system/rep:permissionStore/oak:mount-libs-default`

Oak composite store with Sling / AEM

- Two repositories:
  - `/apps` & `/libs` - stored in a separate, read-only `repository-libs`,
  - other data - stored in the normal repository (Segment, Mongo or RDB).
- The first one has to be created before starting the instance.

Limitations

- Mounts are always read-only
  - Atomic state changes non trivial to get right, fast, scalable
  - Write support requires additional change to multiple Oak subsystems
  - No observation events generated
- Referenceable nodes are not suported in non-default NodeStores
- Versionable nodes are not supported in non-default NodeStores

Ensuring consistency

- **Mount-time checks**
  - No versionable nodes in non-default NodeStores
  - No referenceable nodes in non-default NodeStoress
  - No duplicate entries in unique indexes amongst all NodeStores
  - Node type definitions from mounts consistent with the global node type registry
  - Namespace usage in mounts consistent with the global namespace registry
- **Run-time checks**
  - No cross-mount references may be created at run-time

Application-level changes

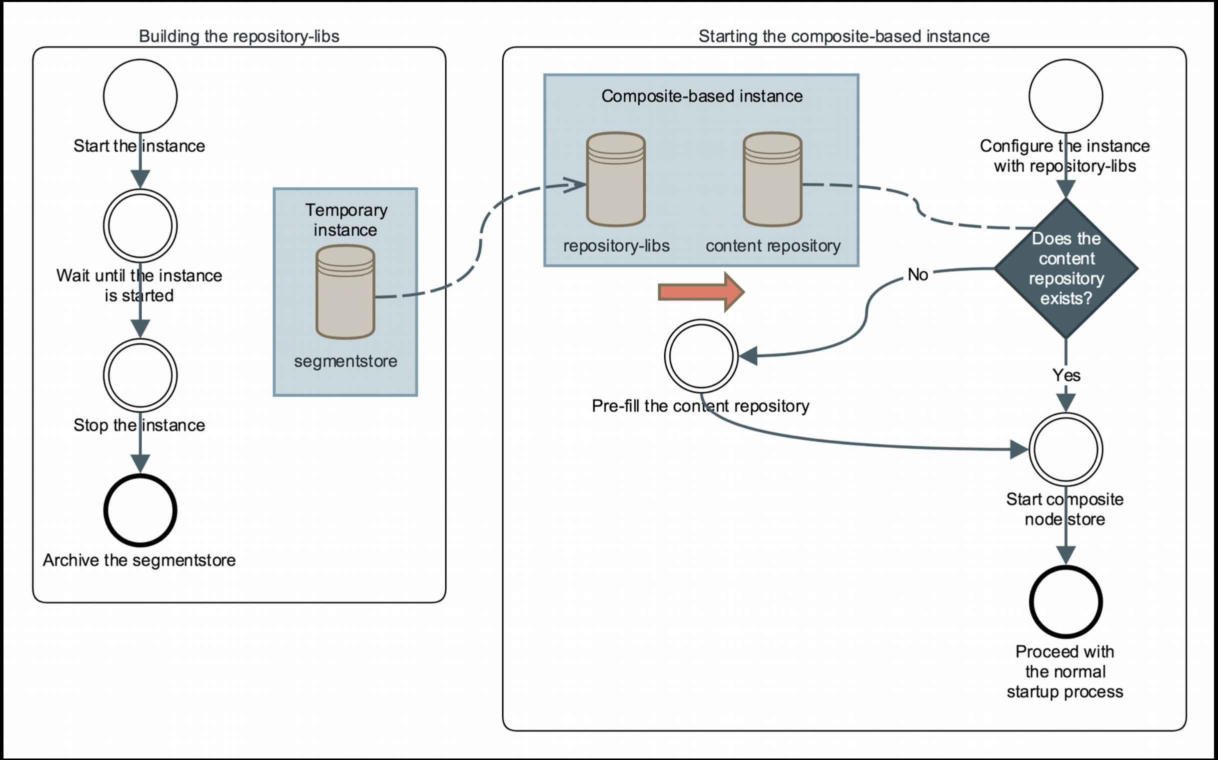
- No run-time changes under /libs or /apps
  - Usually fine or applications
  - More painful for testing
- Read-only status exposed via `Session.hasCapability`, **not** `Session.hasPermission`

Usage

Running instance in the composite mode

- **We need to have two repositories in the composite mode:**
  - one for the application (/apps and /libs),
  - one for the content (everything else).
- **When running the instance in the composite mode, the application part is read-only**
  - so it's not possible to install the application from the content packages, as usual.
- **Therefore, using composite mode involves two-step process:**
  - start instance without the composite node store, to create the application repository,
  - start instance with the composite node store.

Building and starting composite instance


Building the repository-libs
Start the instance
Wait until the instance is started
Stop the instance
Archive the segmentstore
Temporary instance
segmentstore
Starting the composite-based instance
Composite-based instance
repository-libs
content repository
Configure the instance with repository-libs
Does the content repository exists?
No
Yes
Pre-fill the content repository
Start composite node store
Proceed with the normal startup process

Building the composite-enabled instance

- First, the instance is started without the composite node store.
- The composite-init.jar waits until the instance is ready:
  - start level 30,
  - no indexing jobs in progress.
- Then it stops the instance.
- The created `repository` is renamed to `repository-libs`.
- It's a completely initialized repository:
  - `/apps`, `/libs` will be used for the composite node store mount,
  - other content will be used to pre-populate the default node store when running the instance.

Starting the quickstart in composite mode

- The instance is configured with two node stores: the default one and the `repository-libs`.
- They are combined together with composite node store.
- If the instance is started for the first time:
  - the default store is empty, so the composite node store pre-populates it with the content from the `repository-libs`,
  - it copies everything except the apps-related data,
  - once the initial pre-population is done, the instance startup will carry on.
- If the instance references an existing default repository:
  - the startup proceeds, with the new `/apps` and `/libs` part.

Customer application integration

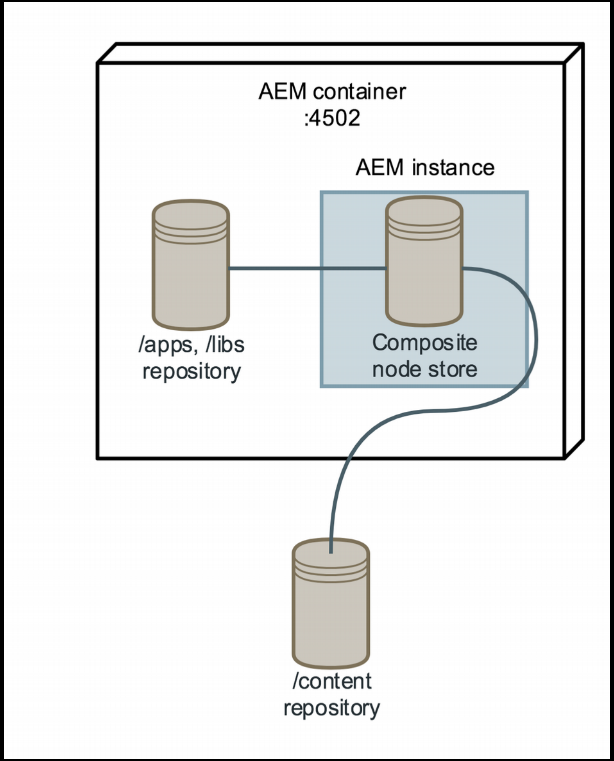
- The customer application can't be installed in the instance runtime.
- It has to be integrated with the Sling/AEM code.
- Sling Provisioning Model:

```
[artifacts startLevel=20]
  com.acme.site/com.acme.site.content/1.0.0/zip

  com.acme.site/com.acme.site.core/1.0.0
  com.acme.site/com.acme.site.email/1.0.0
  com.acme.site/com.acme.site.templating/1.0.0

[configurations]
  com.acme.site.core.AcmeService
    enabled=B"true"
    path="/home/acme"
```

Composite mode with Docker

- This 2-stage process needs to be automated.
- Docker is a perfect tool for the task - it allows to encapsulate both logic and data:
  - `Dockerfile` can be used to orchestrate the required steps,
  - the created Docker image embedds the artifacts and repository-libs.
- A separate image for author and publish.

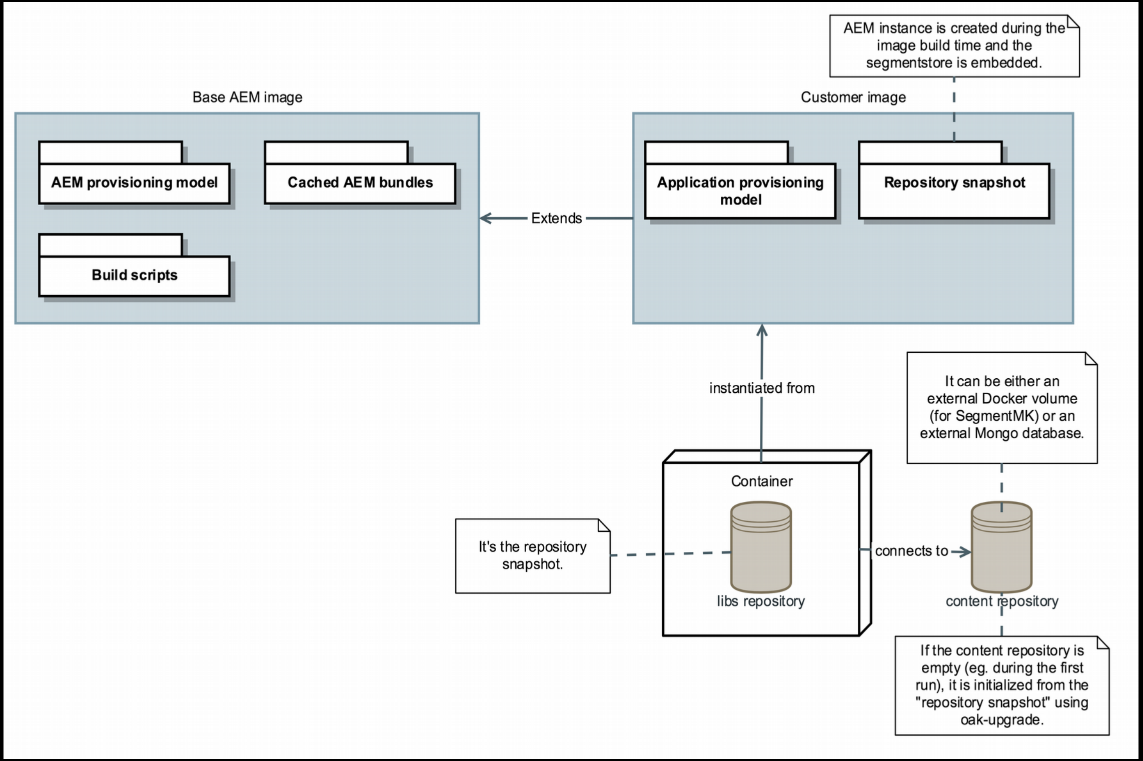
Dockerizing the composite instance



Running the container

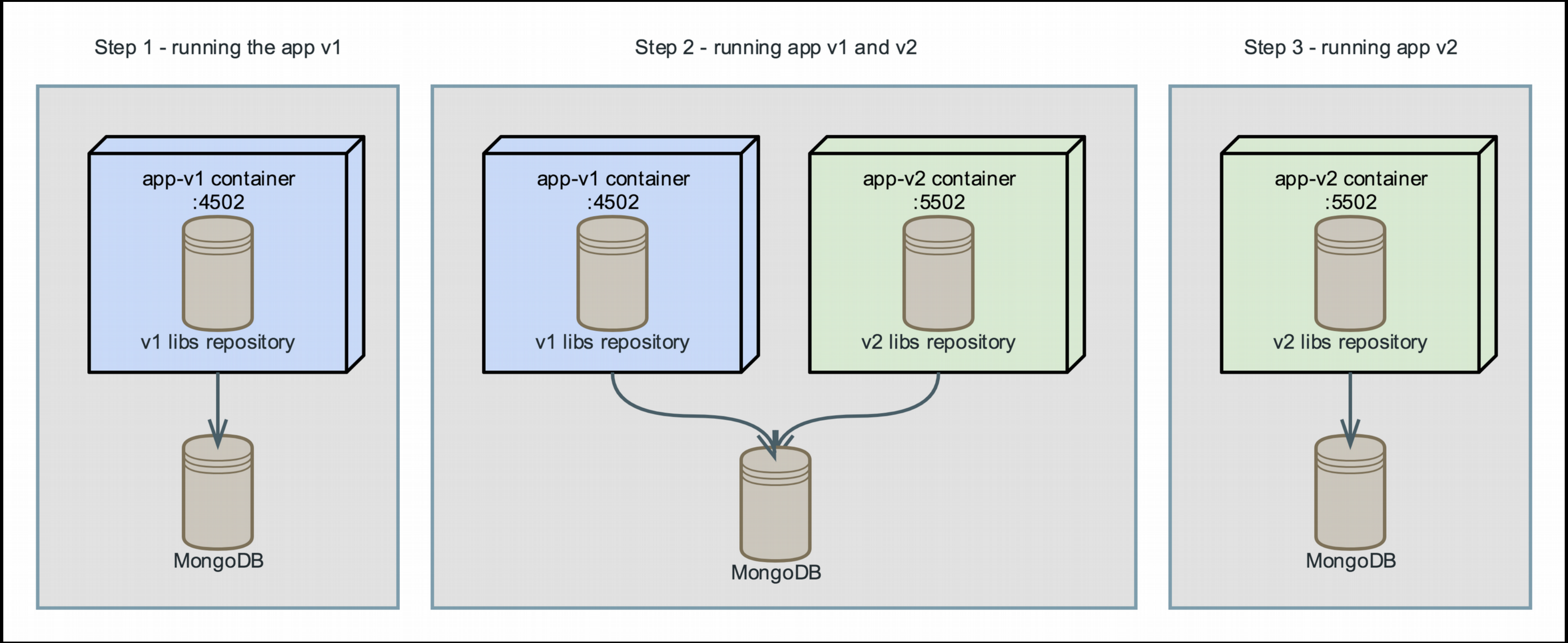
- The container uses an external storage for the non-application content.
- Either `VOLUME` for the TarMK or a Mongo instance.
- The /apps and /libs are served from the embedded repository-libs.

Docker setup summary


AEM instance is created during the image build time and the segmentstore is embedded.
Base AEM image
Customer image
AEM provisioning model
Cached AEM bundles
Build scripts
Application provisioning model
Repository snapshot
Extends
instantiated from
It can be either an external Docker volume (for SegmentMK) or an external Mongo database.
Container
libs repository
connects to
content repository
It's the repository snapshot.
If the content repository is empty (eg. during the first run), it is initialized from the "repository snapshot" using oak-upgrade.

Deployment scenarios

Blue-green deployments



Blue green deployments

- Now we have the whole application code enclosed in the container.
- While the other data (/content) is stored externally.
- This allows to perform a blue-green deployment.
- Blue container is the one running the older version of code.
- Without disabling it, we're creating a green container, running the newer code.
- They are both using the same content, but their `/apps` subtrees and bundles are different.
- Now we can switch the load balancer to point the green container.
- The blue one can be shut down.

Incompatible content changes

- The assumption is that the green container doesn't introduce incomatible changes.
- Otherwise the blue may break.
- In AEM context: eg. no new components should be added if the older version doesn't support them.
- If the property name changes, the new version should fallback to reading the older name as well.
- If the content schema changes, a script may be used to update the content after switching the load balancer,
  - the new application should allow to read the older schema too.

adaptTo()

Demo

Zero downtime demo

- Start a Dockerized, Mongo-based AEM instance with application v1.
- Start the second container, with application v2, connecting to the same MongoDB.
- Confirm it contains a new "video" component.
- Switch the load balancer.
- Destroy the old instance.

adaptTo()

Wrap-up

Resources

- https://jackrabbit.apache.org/oak/docs/nodestore/compositens.html