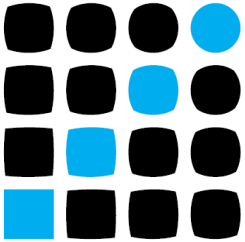


adaptTo()

APACHE SLING & FRIENDS TECH MEETUP

BERLIN, 26-28 SEPTEMBER 2016

AEM Communities and Sling

Siddharth Palaniswami, Adobe

Agenda

- Introduction
- Challenges of UGC and personalized content
- Storage Resource Provider (SRP)
- Social Component Framework (SCF)
- Future

What is AEM Communities

AEM Communities

- Create and manage online communities
- Components for users to engage and collaborate with each other
- Large scale User Generated Content
- Rich, interactive and personalized components

AEM Communities – an Overview

Administration	Site Wizard	Site Templates	Group Templates	Bulk Moderation	In-Context Moderation	Member Mgmt.	Responsive Design	Analytics
Members	Profiles	Scores	Badges	Activities	Notifications	Messages	Social Graph	Social Logins
Functions	Forums	QnA	Blogs	Calendars	Files	NewsFeeds	Groups	Assignment	Catalog
Commons	Comments	Reviews	Ratings	Votes	Tags	Attachments	Search	Translations
Enablement	Resources Management	Learning Paths Management	SCORM Engine	Video Analytics	Progress Reporting	Assignments Reporting
Platform	ASRP/Cloud Storage	JSRP/JCR Storage	MSRP/Mongo Storage	User and Group Sync			

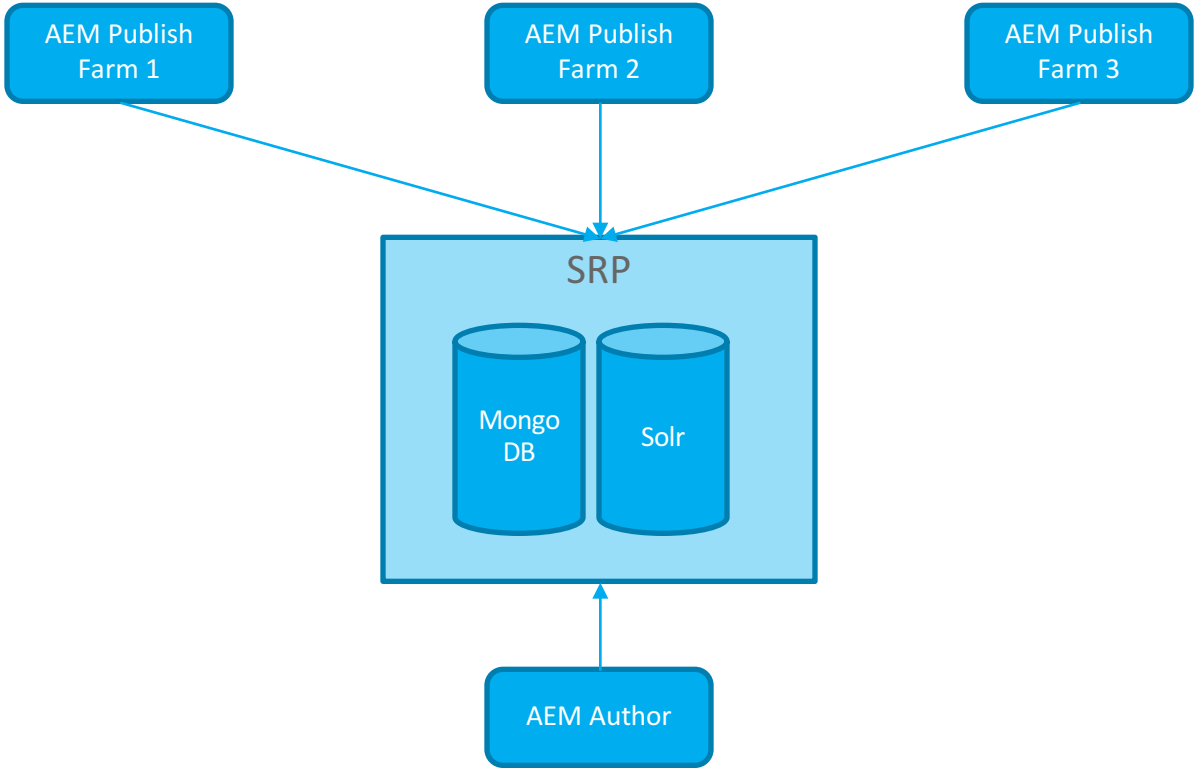
Demo

Challenges with UGC

- **Large volume of uncontrolled writes**
- Content gets created on multiple publish instances and needs to be synced
- Replication became a bottleneck
- Writes and reads need to be fast
- Search needed to scale for high volume of content

Common Store for UGC

Common Store



* MSRP - shown in figure

Storage Resource Provider

- Implements various Sling Resource Provider APIs

```
public interface SocialResourceProvider extends ResourceProvider, ModifyingResourceProvider,
    QueriableResourceProvider {
```

- Provides read, write and search APIs for UGC
- Multiple implementations for various storage options
- Supports atomic increments and pagination
- Search is provided by integrating with Solr

https://docs.adobe.com/docs/en/aem/6-2/develop/communities/essentials/srp.html

Storage Options

- JSRP – UGC stored in local JCR repository
  - Low volume
- ASRP – data is stored via Adobe hosted cloud service
  - Medium volume, managed by Adobe
- MSRP – UGC is sent to a MongoDB setup dedicated for UGC. Solr is used to support search
  - High volume, self managed

Example usage of SRP

- Write to a SRP

```
Map<String, Object> props = new HashMap<String, Object>(5);
props.put("jcr:description", text);
props.put("jcr:date", date);

// use the Resource API to create resource to make this data store agnostic
Resource item;
try {
    item =
        srp.create(resolver, sru.resourceToUGCStoragePath(todolist) + "/" + owner + "/"
                + createUniqueNameHint(text), props);
    resolver.commit();
} catch (final PersistenceException e) {
    LOG.error("Unable to create todo item", e);
    throw new OperationException("Unable to create todo item", 500);
```

Search using SRP

```
final UgcFilter filter = new UgcFilter();
stateGroup.addConstraint(new ValueConstraint<Boolean>("isDone_b", Boolean.TRUE));
filter.and(stateGroup);
filter.and(pathFilters);
filter.addSort(new UgcSort("added", Direction.Asc));
final UgcSearch search = todoListComponent.getResourceResolver().adaptTo(UgcSearch.class);
try {
    SearchResults<Resource> results =
        search.find(null, todoListComponent.getResourceResolver(), filter, 0, 100000, true);
    this.filteredSize = Math.toIntExact(results.getTotalNumberOfResults());
    return results.getResults().iterator();
```

Social Component Framework

Why did we build a framework

- Need for dynamic, rich and interactive components
- User generated content needs to be cached but still personalized
- Search engine friendly – community content discovery is largely via search engines
- Reduce complexity and time to customize and extend components
  - Customizing OOTB components to fit customer's design should be easy
  - Extending functionality should not mean large projects
  - Should be very easy to customize look and feel and UX
  - Need for re-usability
- Integrate with legacy systems or other custom applications
  - Inherently support a HTTP API

What does our component framework do?

Provides a **collection of services, APIs and patterns** that enable developers to create dynamic components. Components built using the framework are easy to **customize**, **extend** and **reuse**.

- Skinnable
- Simple/Reusable Templates
- HTTP API
- Server Side Extensibility
- Dynamic Content
- Client Side Extensibility
- SEO Friendly
- Client Side Rendering

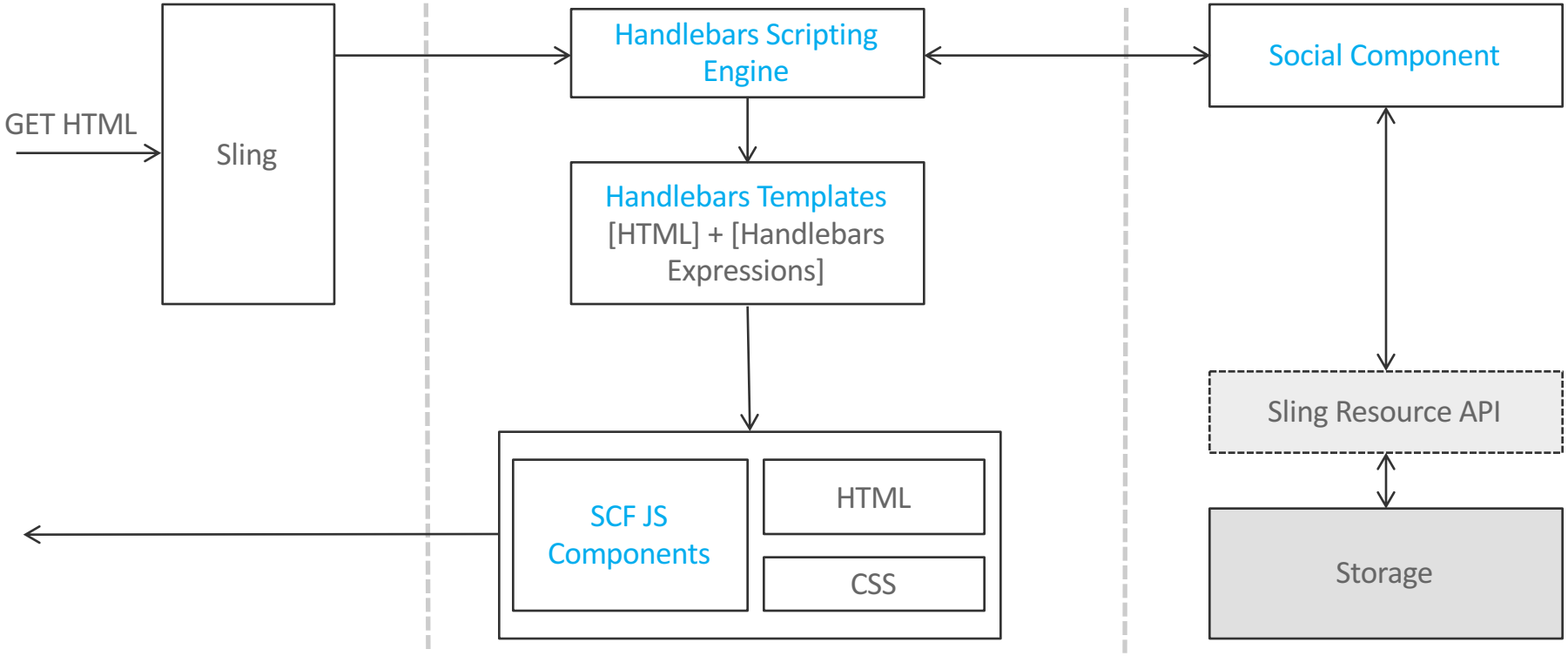
Demo

Code Example

Illustrates how to build a SocialComponent that reads/writes/searches UGC using the SRP storage options

https://github.com/Adobe-Marketing-Cloud/aem-communities-todomvc-sample

GET Request


GET HTML
Sling
Handlebars Scripting Engine
Handlebars Templates
[HTML] + [Handlebars Expressions]
SCF JS Components
HTML
CSS
Social Component
Sling Resource API
Storage

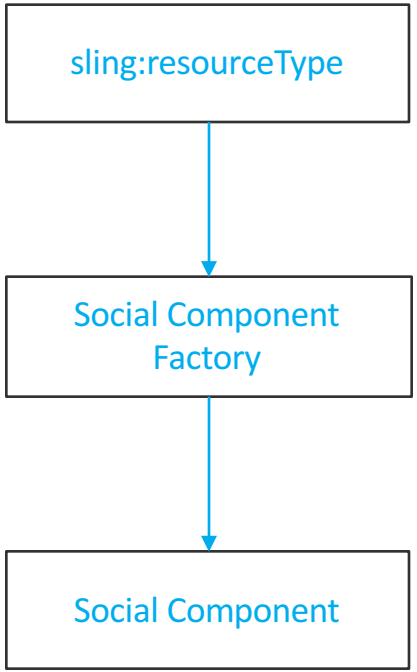
Example SocialComponent

```java
/**
 * A Social Component that represents a single todo item
 */
public interface TodoItem extends SocialComponent {

    /**
     * @return the text of the todo item
     */
    String getItemText();

    /**
     * @return true if the item is still pending, false if the item has been completed
     */
    boolean isActive();
}

public class TodoItemImpl extends BaseSocialComponent implements TodoItem {

    private final ValueMap itemProps;

    public TodoItemImpl(Resource resource, ClientUtilities clientUtils) {
        super(resource, clientUtils);
        itemProps = resource.adaptTo(ValueMap.class);
    }

    @Override
    public String getItemText() {
        return itemProps.get("jcr:description", "");
```

Social Component

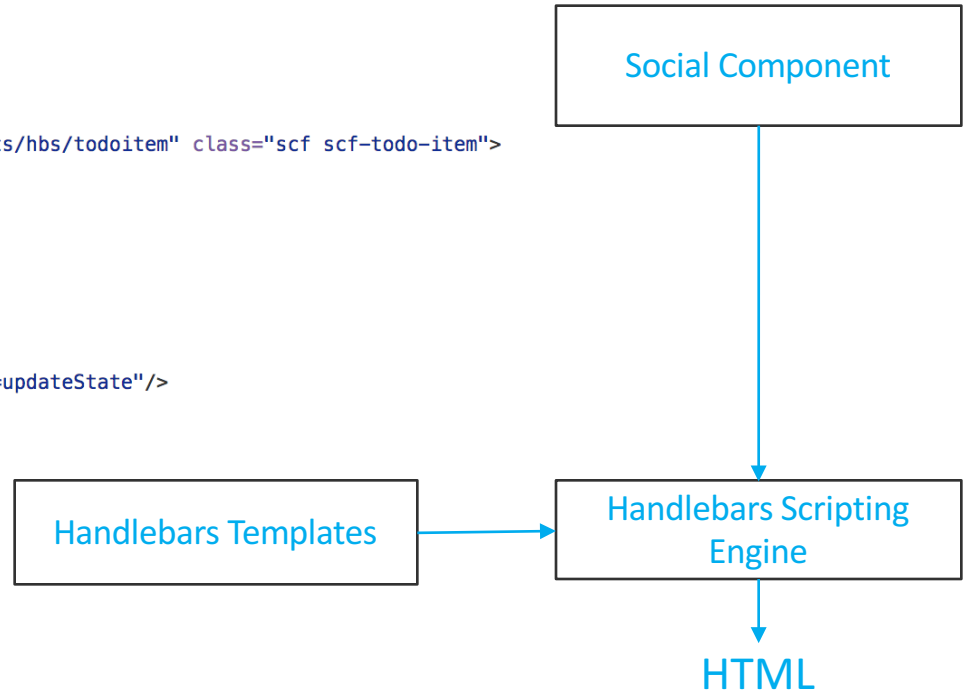
Registering a SocialComponent

```
@Service
@Component
public class TodoItemFactory extends AbstractSocialComponentFactory implements SocialComponentFactory {

    @Override
    public SocialComponent getSocialComponent(final Resource item) {
        return new TodoItemImpl(item, getClientUtilities(item.getResourceResolver()));
    }

    @Override
    public String getSupportedResourceType() {
        return "scf-todo/components/hbs/todoitem";
    }
}
```

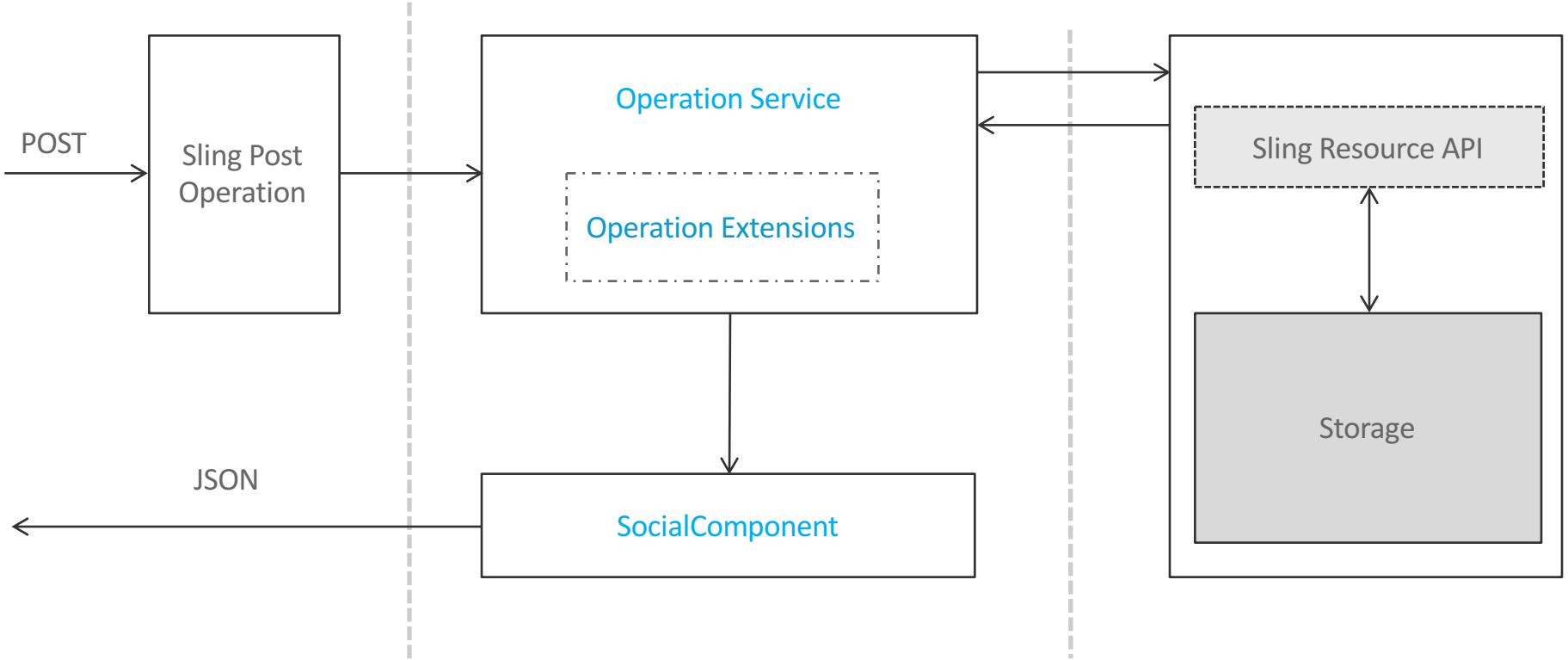


Handlebars Template

```
<li data-component-id="{{id}}" data-scf-component="scf-todo/components/hbs/todoitem" class="scf scf-todo-item">
    {{#if active}}
        <div class="scf-todo-item-content">
            <input type="checkbox" evt="change=updateState"/>
            <span>{{itemText}}</span>
        </div>
    {{else}}
            <div class="scf-todo-item-content scf-is-done">
                <input type="checkbox" checked="checked" evt="change=updateState"/>
                <span>{{itemText}}</span>
            </div>
    {{/if}}
</li>
```



POST Request



SocialComponents as RESTful endpoints


```
{
  "id": "/content/usergenerated/asi/mongo/content/todos/jcr:content/todos/jdoe@geometrixx.info/wFox4e-my-very-first-todo-t",
▶ "properties": {…}, // 9 items
  "itemText": "my very first todo to get done",
  "active": true,
  "resourceType": "scf-todo/components/hbs/todoitem",
  "url": "/content/usergenerated/asi/mongo/content/todos/jcr:content/todos/jdoe@geometrixx.info/wFox4e-my-very-first-todo-t.social.json"
}
```

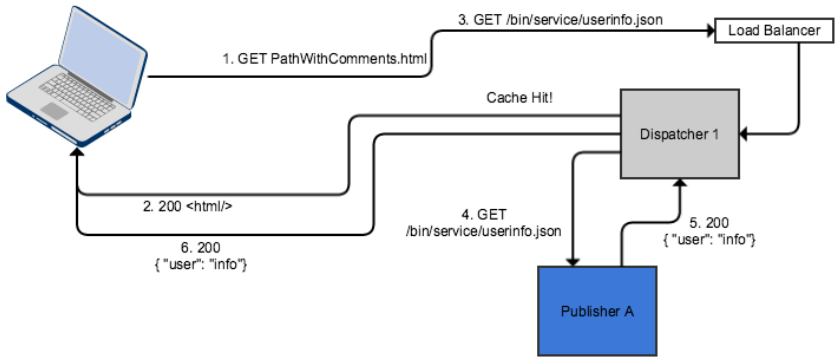

adaptTo()

Customizing/Extending OOTB Components

- Skinning
  - Only need to change the CSS
- Look and Feel and UX
  - Change template, CSS and extend/override JavaScript Models and Views
- Make more/less information available to the template or to the GET endpoint
  - Extend the OOTB SocialComponent(only need to add/subtract what you need)
  - Register custom SocialComponent for component resourceType
- Add some custom processing during operations
  - Write an OperationExtension
  - Listen for OSGi events
- Add a new custom operation
  - Create new Sling Post Operation
  - Use existing Operation Services as needed
- Integrating with 3rd party services
  - Use HTTP endpoints for Create/Edit/Delete
  - Use /path/to/resource.social.json to read and present resources

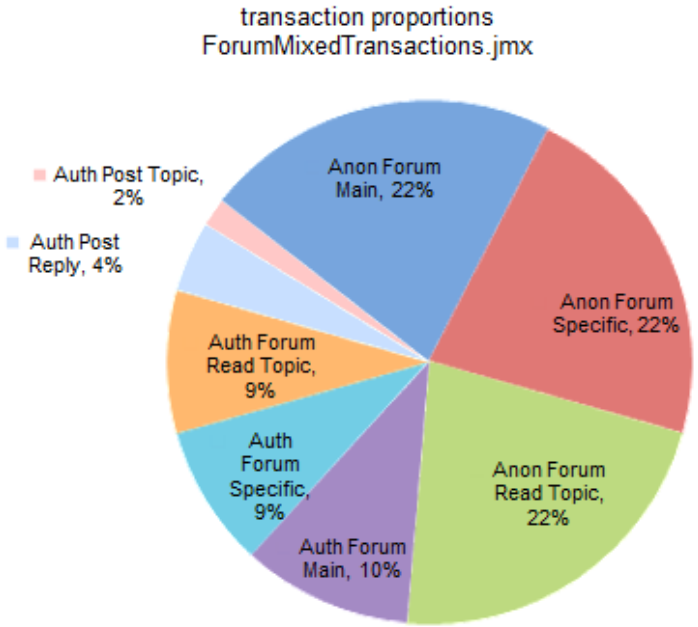
Caching and Dynamic Content

Caching and Personalizing views



- Server rendered component is cached for anonymous users
- Components re-rendered on client side after side loading user specific information
- Caches are flushed by listening for OSGi events that update UGC

SCF+SRP Performance



- JCR with Replication + Fully server rendered components (JSP)
  - CRX2
  - 2 node active/passive cluster
  - **5 transactions per second**
- SRP - SCF for components + MSRP (Mongo + Solr) for UGC
  - OAK (for content and component nodes)
  - 4 node publish farm
  - Dispatcher
  - 100 transactions per second

What's next?

What's next?

- SCF and SlingModels
  - resourceType to SlingModel binding - SLING-5992
  - "export" JSON serialization of SlingModel (.export.json)
- SCF Next
  - Extend SlingModels
  - Optimized JSON views
  - Opensource SCF?

Thank You