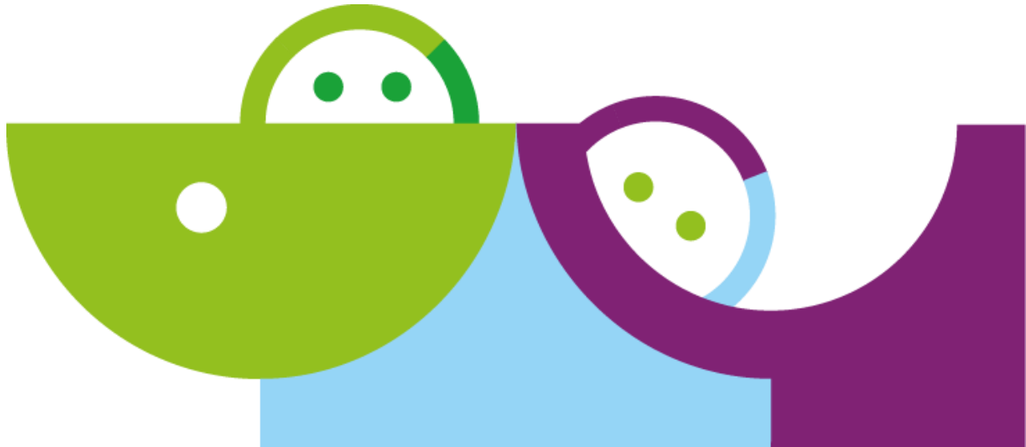
Meet the Lab
June 17th – 18th 2015

IBM Lab Böblingen



Please Note

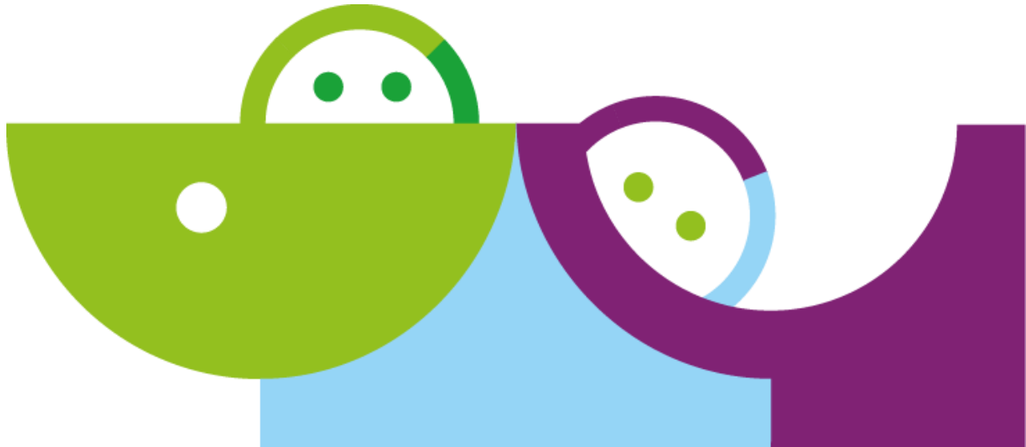
- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion
- Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

New Aspects of the Portlet Programming Model:

Client Side and The Cloud

Dr. Carsten Leue

Portal Architect



Agenda

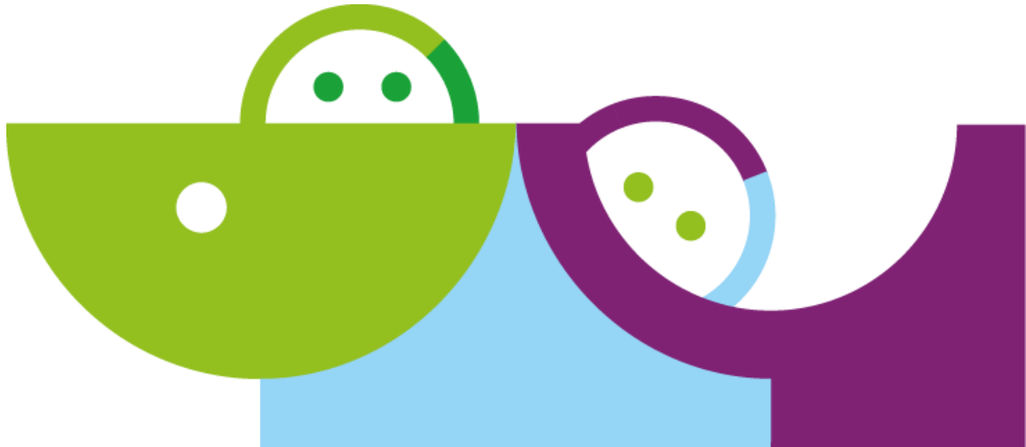
- Introduction
- Client Side Portlet Programming
  - Upcoming Technologies
  - How to get the best synergy between Portlets and Scripting
- Portlet Deployments in the Cloud
  - Use cases and topologies
  - How it all fits together
- **Target Audience**
  - Developers, Application Designers, Architects

Motivation

- UI Technologies change rapidly – and so do Customer expectations
  - Bootstrap, Angular-JS, Polymer, …
  - How can applications make use of these new techniques while still collaborating with existing assets?
  - How to combine application using different client side technologies?
  - Is the **Portlet Programming Model** flexible enough to cope with today's challenges?
- The "Cloud" offers new deployment opportunities
  - Extended reuse by Software-as-a-Service offerings
  - How can this be combined with existing assets?
  - How can cloud based services be used while still maintaining sensitive data on Premise?

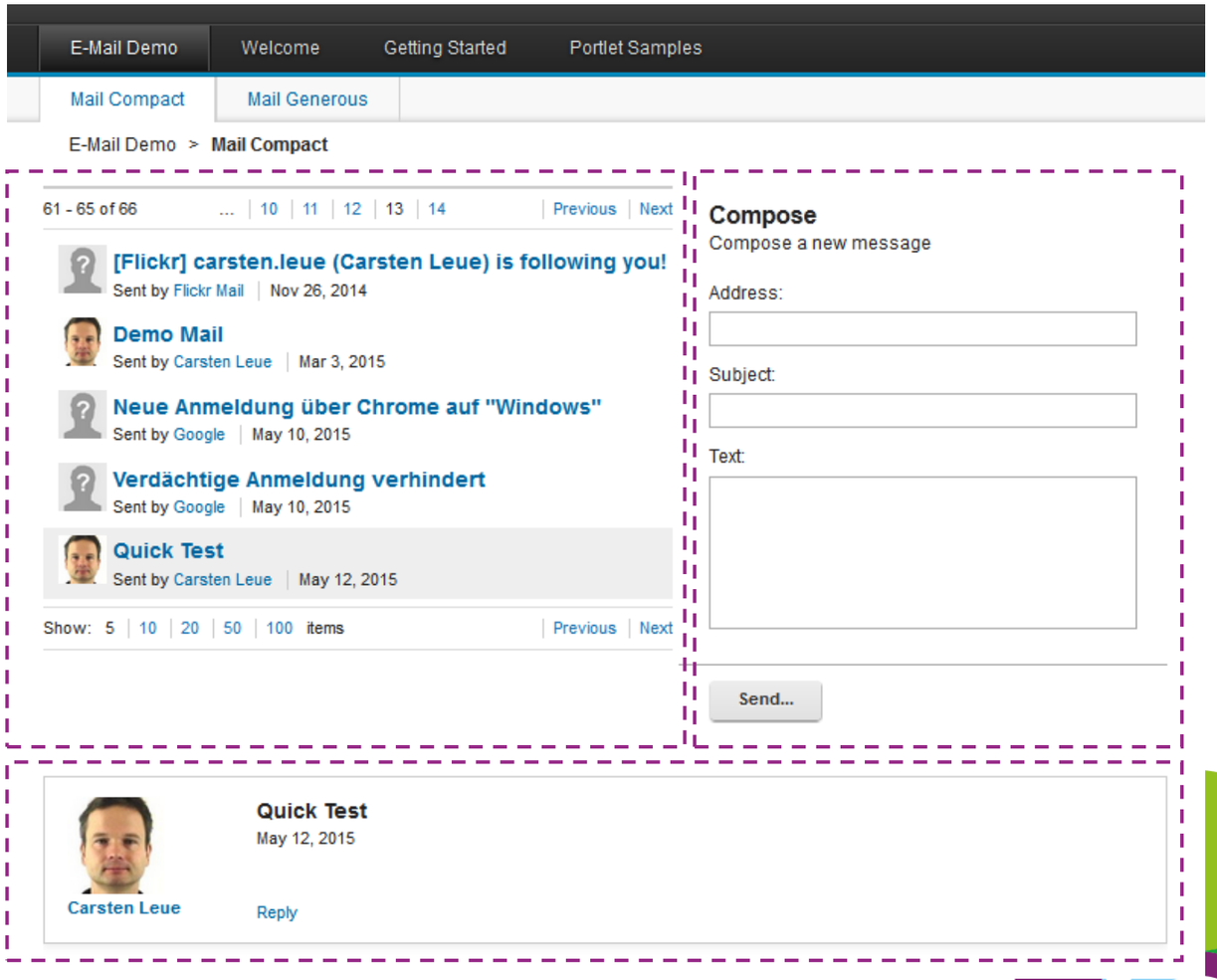
Programming Model Patterns

Focus on Client Side Portlet Programming



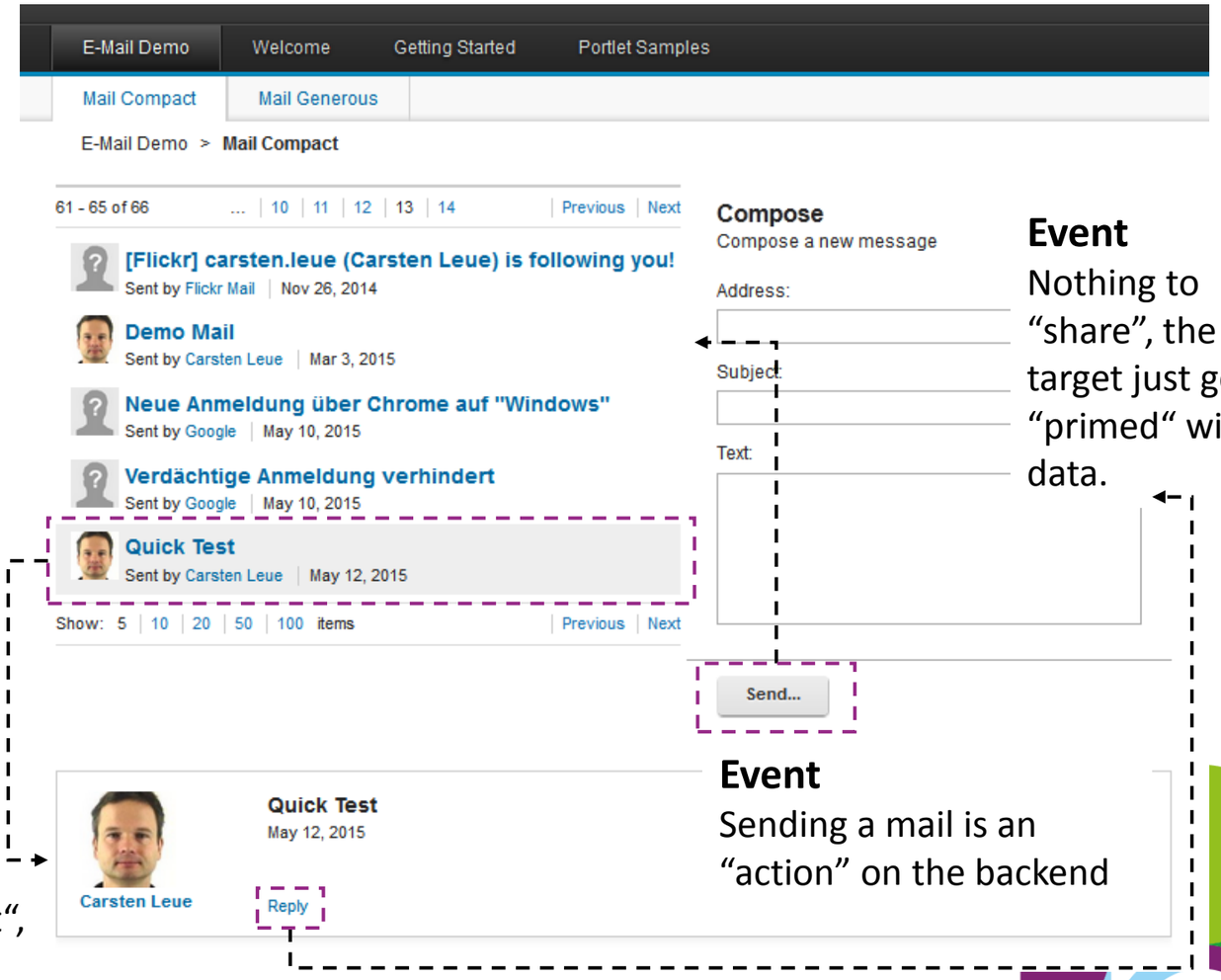
Componentization

- Portlets are the primitive UI building blocks of Choice
  - JSR 286 defines Java API
  - UI technology is a choice of the developer (JSP, JSF, …)
  - Run locally (Java) or remotely (WSRP)
- Applications are built by combining portlets on pages
  - Same components - different layouts
  - Coordination of state between components by container



Coordination

- Portlets communicate with the runtime and with each other
  - Independent of UI frameworks
  - Mediated by the portlet container
- Key concept: **state handling**
  - Private render parameters
  - **Public render parameters**
  - Portlet Eventing



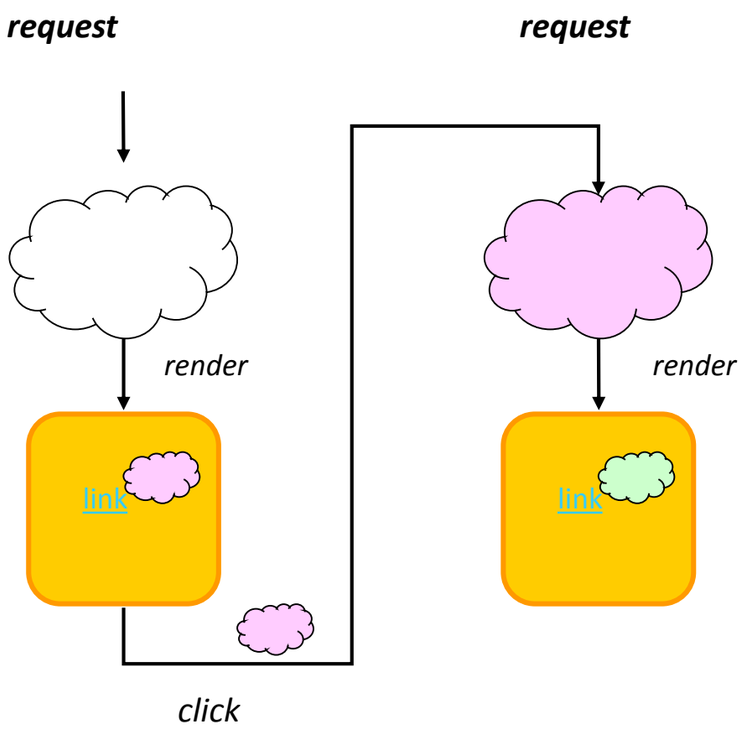
Event
Nothing to "share", the target just gets "primed" with data.

Event
Sending a mail is an "action" on the backend

Public Render Parameter
Selected message is only context", no server side state changes.

Coordination - Discussion

- State coordination uses the URL to maintain state
  - → state changes require changes to the URL
  - → full page refreshes
- AJAX support in portlets is limited to fetching data
  - No AJAX based state change supported in JSR 286
- This limitation makes it difficult to interface with cool **client side** libraries
  - Angular JS
  - Polymer
  - CSS3 Transitions
  - …

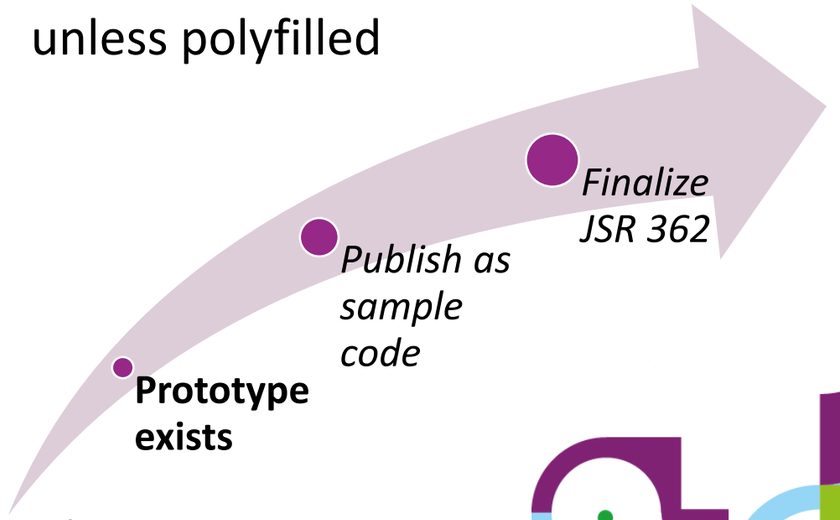


Typical Request Flow
Each state change is communicated via a separate request.

Stateful AJAX Support – JSR 362

COMING SOON!

- State handling concept remains identical to JSR 286
  - Based on Private and Public Render Parameters
- State changes without full page refreshes
  - Client side JavaScript Library “PortletHub” maintains and coordinates state
  - Portlets participate by registering JS call-backs
- Actions and Events remain server side
  - No page refresh required
  - Business logic remains encapsulated on the server
- Asynchronous Markup refreshes
  - built on top of the **existing resource serving**
- Sample Implementation for WebSphere Portal
  - Full support for back and forward button
  - Bookmarkable URLs
  - Works also via WSRP!
- Limitations
  - Requires A+ Promises, so no IE support, unless polyfilled



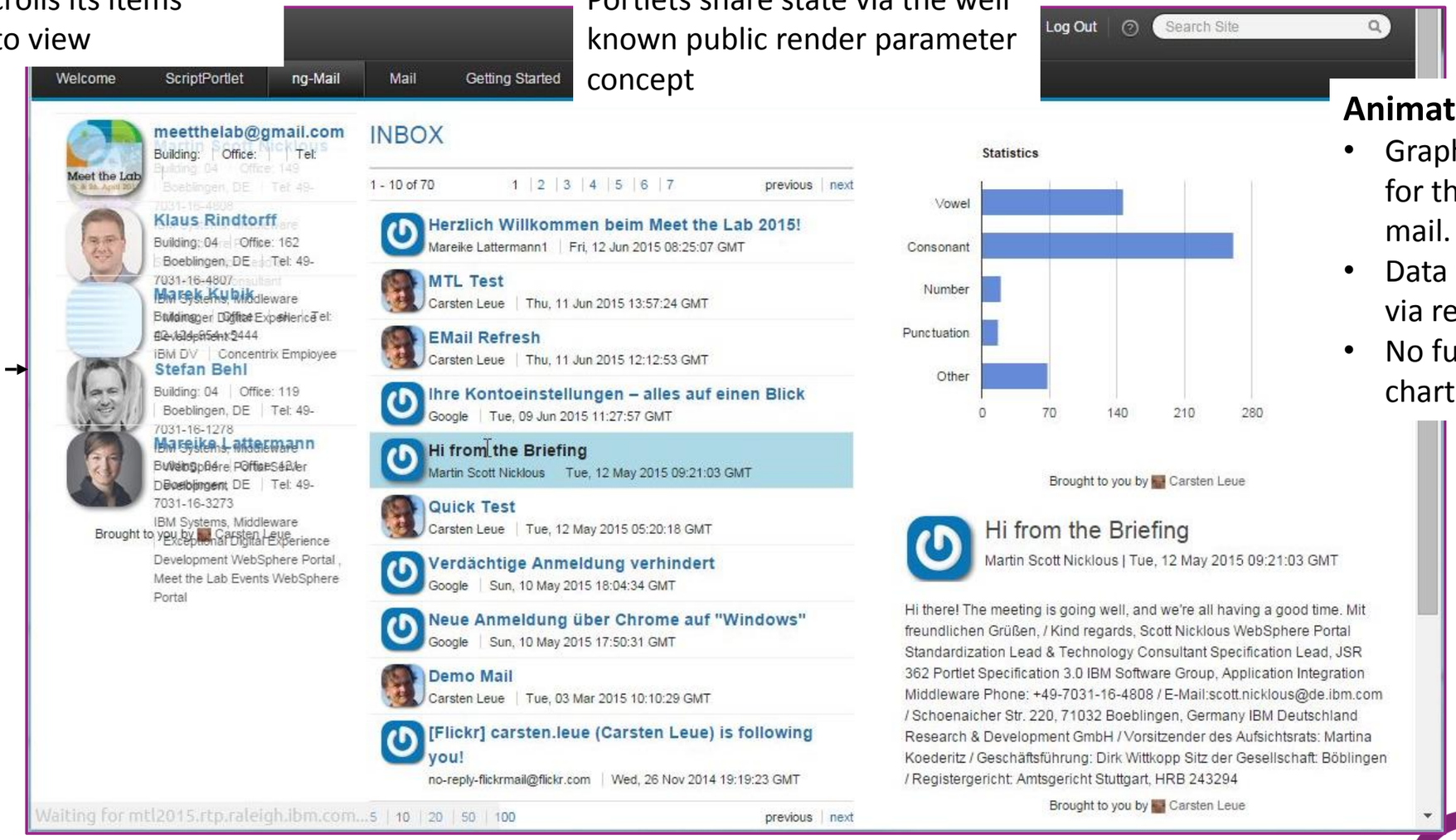
Sample Application – Smooth Updates

Animation

Person list scrolls its items **smoothly** into view

Public Render Parameters

Portlets share state via the well known public render parameter concept



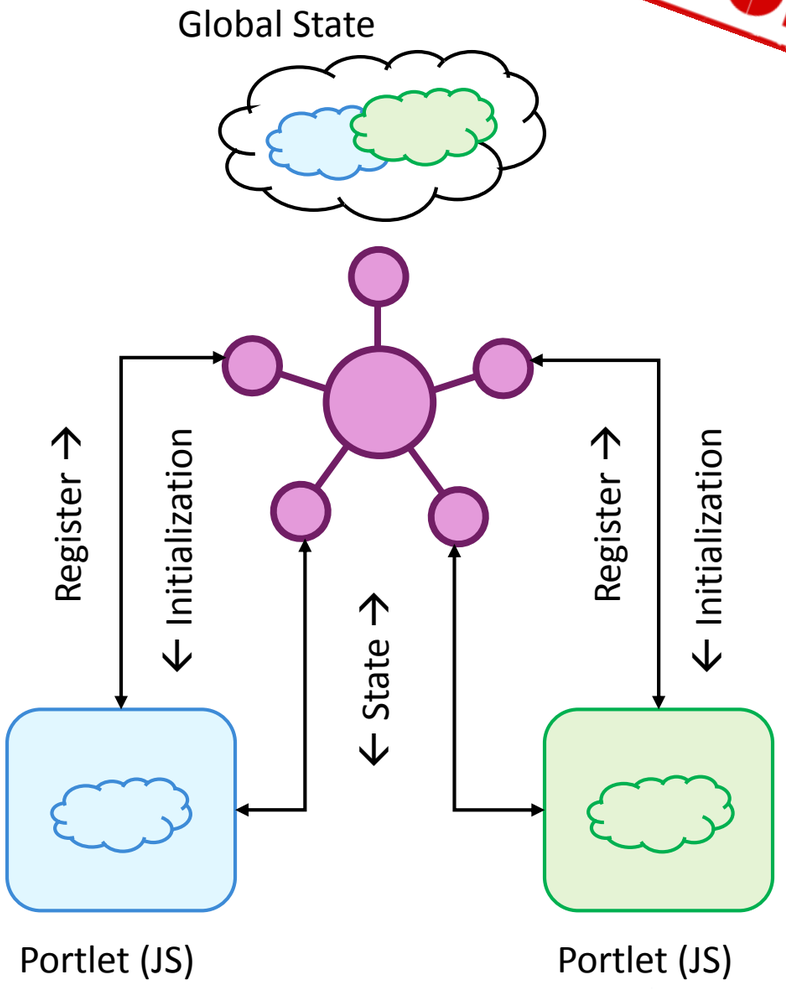
Animation

- Graph represents statistics for the currently selected mail.
- Data fetched asynchronously via resource serving
- No full page refresh involved, chart updates **smoothly**

Portlet Hub – Easy-to-Use but Powerful

COMING SOON!

- Bootstrap
  - A Portlet registers a call-back to a global Portlet Hub library instance
  - Inline or via an **onLoad** handler
- Initialization
  - The Portlet Hub notifies the portlet and passes an API stub when the initialization is complete
- State Changes
  - State changes are propagated from the Hub to the portlet (public and private state)
  - The portlet pushes state changes back to the Hub
- Actions and Events
  - Actions and events are executed on the server, asynchronously
  - Resulting state changes are communicated via state change events



Portlet Hub – Bootstrap

COMING SOON!

Portlet JSP

```
<%@ page session="false" buffer="none"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="portlet" uri="http://java.sun.com/portlet_2_0"%>

<portlet:defineObjects/>
<c:set var="ns" value="${renderResponse.namespace}"/>
<div id="${ns}root" data-ns="${ns}">
  ...
</div>

<script type="text/javascript">
my_library_init(${ns}root);
</script>
```

Portlet Hub API

The hub passes an instance-scoped API call-back to the initialization method.

Portlet (JS)

```
function my_library_init(aRoot) {

/**
 * Callback invoked if the initialization of the
 * portlet hub is complete
 */
function xInit(hubAPI) {
// ...
}

/**
 * Place the register callback
 */
return portlet.register(aRoot.dataset.ns).then(xInit);
};
```

Registration

Use the namespace to identify the portlet instance.

All asynchronous operations are based on **A+ Promises** (ES6).

Portlet Hub – Initialization

COMING SOON!

Portlet (JS)

State Call-back Handler

Called whenever state changes.

- After the page load, to initialize the portlet
- After a portlet changes its private or public render parameters
- After the public render parameter context changes for any other reason

State Call-back Registration

Register a call-back to receive notifications about new state.

The call-back is scoped to the instance.

```
function xInit(hubAPI) {

/**
 * The current state document
 */
var xState;

/**
 * Callback function invoked by the portlet hub whenever state changes.
 */
function xOnStateChange(aType, aState) {
  // remember the state
  xState = aState;

  // update the UI accordingly
}

/**
 * Callback of the portlet hub that notifies the controller about
 * modifications to its state
 */
hubAPI.addEventListener("portlet.onStateChange", xOnStateChange);
}
```

Portlet Hub – State Changes

COMING SOON!

State Call-back Handler

- Receives new state and updates the required portions of the UI.
- **Typical use case:** use REST service to fetch UI fragments via serveResource

```
/**
 * Callback function invoked by the portlet
 * hub whenever state changes.
 */
function xOnStateChange(aType, aState) {
  // remember the state
  xState = aState;

  /**
   * Fetch the markup update. Note that the
   * resource URL will automatically contain
   * the current state
   */
  hubAPI.createResourceURL({}, "cacheLevelPortlet")
    .then(xhrGet)
    .then(xUpdate);
}
```

```
/**
 * Event handler associated with our link
 */
function xOnClick(aEvent) {
  var e = aEvent || xWindow.event;

  // change the state
  var newState = xState.clone();
  newState.parameters.key = [ "new value" ];
  /**
   * Communicate the new state to the hub,
   * this will cause a callback
   * to the state-change-callback
   */
  hubAPI.setPortletState(newState);
  // indicate that we handled the click
  e.preventDefault();
}
```

State Change Trigger

In an **`onClick`** handler:

- Clone current state
- Update the clone to reflect the new state
- Communicate this state change

Portlet Hub – Actions and Events

COMING SOON!

Action Trigger

- Portlet triggers an action as an asynchronous call
- No other actions can be triggered in parallel
- The Portlet Hub will invoke the action phase on the server, including JSR 286 event distribution
- After the action, all client side portlets receive their state changes. **State change required!**

```
/**
 * Triggers an action by sending a form to the
 * portlet
 */
function xOnAction(aForm) {
// sends the content of the form as an action
portletHub.action(aForm);
}
```

- Actions and Events are per definition „unsafe“ operations
  - Meant to execute business logic and update server side state
  - Consequently there is no way to receive an action or event on the client
- Client-side Events
  - Meant to capture user-interaction events (e.g. keyboard, mouse)
  - Not comparable to JSR 286 events
  - Portlet can subscribe to any DOM handler, not special Portlet Hub support required
  - Coordination across portlets via „onStateChanged“ events

Why this Hybrid Client-Server Approach?

- **Goal:** Leverage the Good Parts of JavaScript, avoid the Bloat
- Minimize the required Script footprint
  - At a minimum, only DOM update logic has to be implemented in JavaScript
  - The generation of updated markup fragments can be kept on the server (via *serveResource*)
  - No need to replicate the numerous portal features as a JavaScript API
    - All portal APIs are available in the server side part and can be funnelled through serveResource
- Separation of Concerns
  - JavaScript adds dynamic aspects to the UI
  - Business logic is best kept on the server
- Information Disclosure
  - JavaScript code is visible to every end-user via the browser
  - Implementing important business logic on the server makes sure not to disclose undesired details
- First-Page Experience
  - State is available also at page refresh time
  - The Portlet can implement its initial view based on this state

Why use Portlets instead of Full Page Apps?

- It looks tempting to implement my application as a monolithic JavaScript application
  - Component reuse at build time possible
  - Why the extra complexity to use portlets?
- Life cycle and Maintenance
  - Even if started at the same time, pieces of the project tend to have different life cycles
  - Portlets allow each component to be developed at its own pace
- UI framework of Choice
  - UI frameworks tend to change over time
  - With portlets, each application piece can use different frameworks and still work together (e.g. if some parts cannot be updated in time)
- Proper separation of Concerns
  - JavaScript based UIs require backend support in form of REST services
  - Portlets allow to provide REST services (via serveResource) and UI as a consistent piece of work
  - REST services can be tailored and optimized to the needs of the client side layer

Key Concept: Resource Serving

- JSR 286 introduces **ResourceURLs** to support Web 2.0 use-cases in a portlet
  - Resource URLs trigger the invocation of the serveResource method of the portlet
  - A portlet can produce markup fragments or data within the portal context
    - Markup fragments can contain URLs
    - Access to navigational state, portlet mode, window state, session, preferences …
  - Resource requests support different types of caching:
    - **FULL**: no access to the navigational state of the portlet, only to session and preferences
    - **PORTLET**: access to the navigational state of the portlet, but NOT to the state of the portal
    - **PAGE**: full access to all information
- **When to use**
  - See next slides for use-cases
- **When NOT to use**
  - For static resources shipped with the portlet, use

```
response.encodeURL(request.getContextPath() + resourcePath)
```

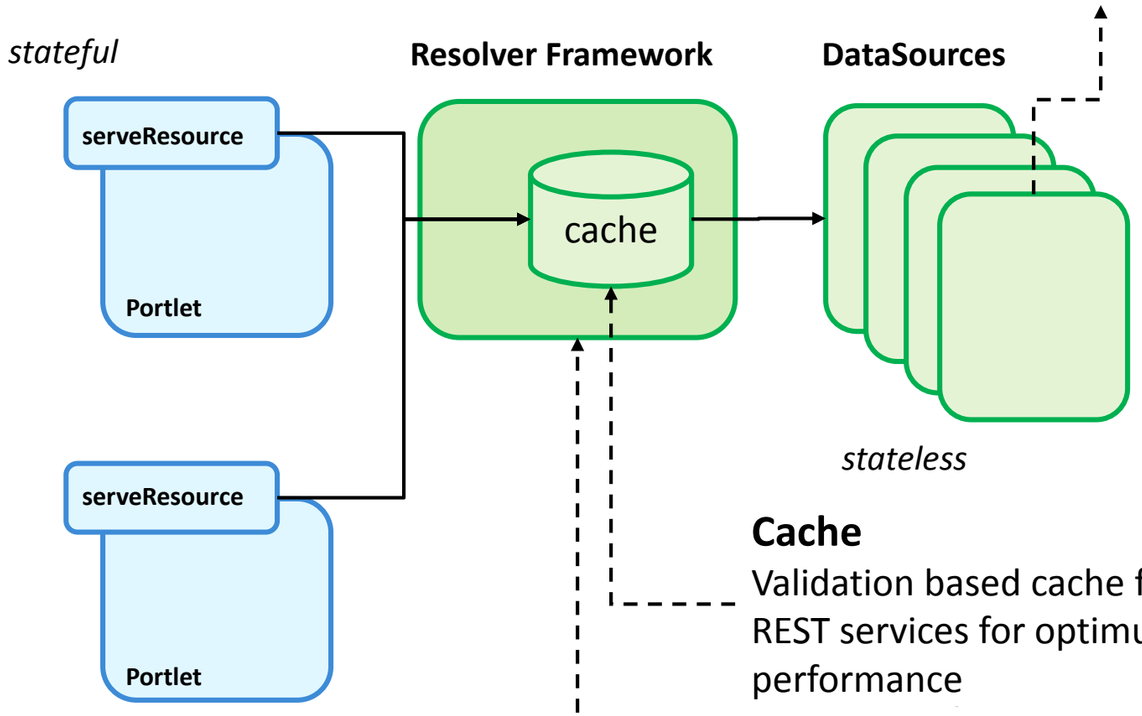
Resource Serving and Portlet Hub

- Context sensitive REST entry
  - Runs in the portlet context
  - Access to public and private render parameters and preferences
  - Can produce markup or any other mime-type
- Ideal mechanism to provision client side markup updates
  - Either via rendered markup fragments
  - Or via a context sensitive data feed, converted into markup on the client
  - Complex logic can reside on the server
- Fully cacheable
  - Scope and lifetime can be set programmatically
- Works via WSRP!
- New with Portlet Hub
  - Resource URLs can be computed on the **client side**

Resource Serving and REST Services

- Resource serving
  - Portlet specific data entry
  - Typically produces data that is tailored to the needs of the script layer
  - Not meant to be shared across portlets
- Reuse by delegating to the server side service stack
  - "serveResource" augments the request with portlet specific aspects (parameters and preferences) and delegates to the **stateless** service layer
  - The result is transformed to meet the needs of the client side script layer
- WebSphere Portal's **Resolver Framework** is a good fit for the server side service stack

DataSources
Implementation of REST services, with strong support for XML, JSON, text, etc



Cache
Validation based cache for REST services for optimum performance

Resolver Framework
Framework for the registration and execution of server side REST services.

Pulling-in JavaScript Libraries

- Client side portlets often require additional libraries or CSS
  - Angular JS
  - JQuery
  - …
- **New since 8.5 CF03**: Resource Aggregation for Portlets
  - Portlets declare their module dependencies in the portlet deployment descriptor
  - At page aggregation time, the **system combines** all required resources, automatically
- Key advantages
  - Optimization of the number of request required to load the dependencies
  - Portlets are **self contained** and no longer depend on the page admin to set the correct profile
  - Makes the overall system simpler to use and efficient
- Per Theme setting
  - Enable portlet level preferences via **Theme Metadata**

portlet.xml

```
<portlet-preferences>
     <!-- indicate that we depend on Angular -->
     <preference>
          <name>capability.2.id</name>
          <value>angular</value>
          <read-only>true</read-only>
     </preference>
     <preference>
          <name>capability.2.minValue</name>
          <value>1.3</value>
          <read-only>true</read-only>
     </preference>
     <!-- indicate that we depend on the portlet hub -->
     <preference>
          <name>capability.3.id</name>
          <value>portlethub</value>
          <read-only>true</read-only>
     </preference>
     <preference>
          <name>capability.3.minValue</name>
          <value>0.1</value>
          <read-only>true</read-only>
     </preference>
     <!-- if unavailable we'd fail than fall back -->
     <preference>
          <name>capabilities.selfManaged</name>
          <value>false</value>
          <read-only>true</read-only>
     </preference>
</portlet-preferences>
```

Pulling-in JavaScript Libraries – Tips and Tricks

- Manage the client side resources of the Portlet **as a Module**
  - Supports both CSS and JS
  - The portlet module can declare dependencies on other modules
  - All script code for the page and all portlets gets aggregated together automatically
- Code Version vs. Portlet Namespace
  - Script code should be authored as a client side module with a unique identifier
  - Each portlet might need to get its own instance of the library in its own namespace
  - The Library should be **stateless**, the instance can be **stateful**

plugin.xml

```
<extension
point="com.ibm.portal.resourceaggregator.module"
id="com_ibm_cleue_mail_graph_angular_module" >
<module id="com_ibm_cleue_mail_graph_angular_module">
  <capability id="com_ibm_cleue_mail_graph_angular"
value="0.1"/>
  <prereq id="com_ibm_cleue_mail_module" />
  <prereq id="portlethub" />
  <prereq id="googleloader" />
  <prereq id="angular" />
  <prereq id="wp_client_ext" />

  <contribution type="head" >
  <sub-contribution type="js">
  <uri value="res:{war:context-
root}/resources/com/ibm/wps/cleue/mail/angular/graph/sc
ript/script.js" />
  </sub-contribution>
  <sub-contribution type="css">
  <uri value="res:{war:context-
root}/resources/com/ibm/wps/cleue/mail/angular/graph/st
yles/styles.css" />
  </sub-contributio
  </contribution>
</module>
</extension>
```

Portlet Module

- Defines its own capability, so multiple portlets load a resource only once
- Dependencies on prerequisites are resolved automatically

Pulling-in JavaScript Libraries – Tips and Tricks

Portlet (JS)

```
/**
 * Generate a closure for the library
 */
(function() {
  /**
   * Stateless functions come here
   */
  function xStateless() {
   alert("hello");
  }
  /**
   * Stateful instances scoped to a portlet come here
   */
  function xCreate(aRoot) {

    var xCounter;

    function xStateful() {
     xCounter++;
    }
    // register the API methods to
    instance
    aRoot.increment = xStateful;
    aRoot.alert= xStateless;
  }
  /**
   * Register the global entry into the library
   */
  window.myAPIv1 = xCreate;
}());
```

Instance Scope
Variable scoped to the portlet instance

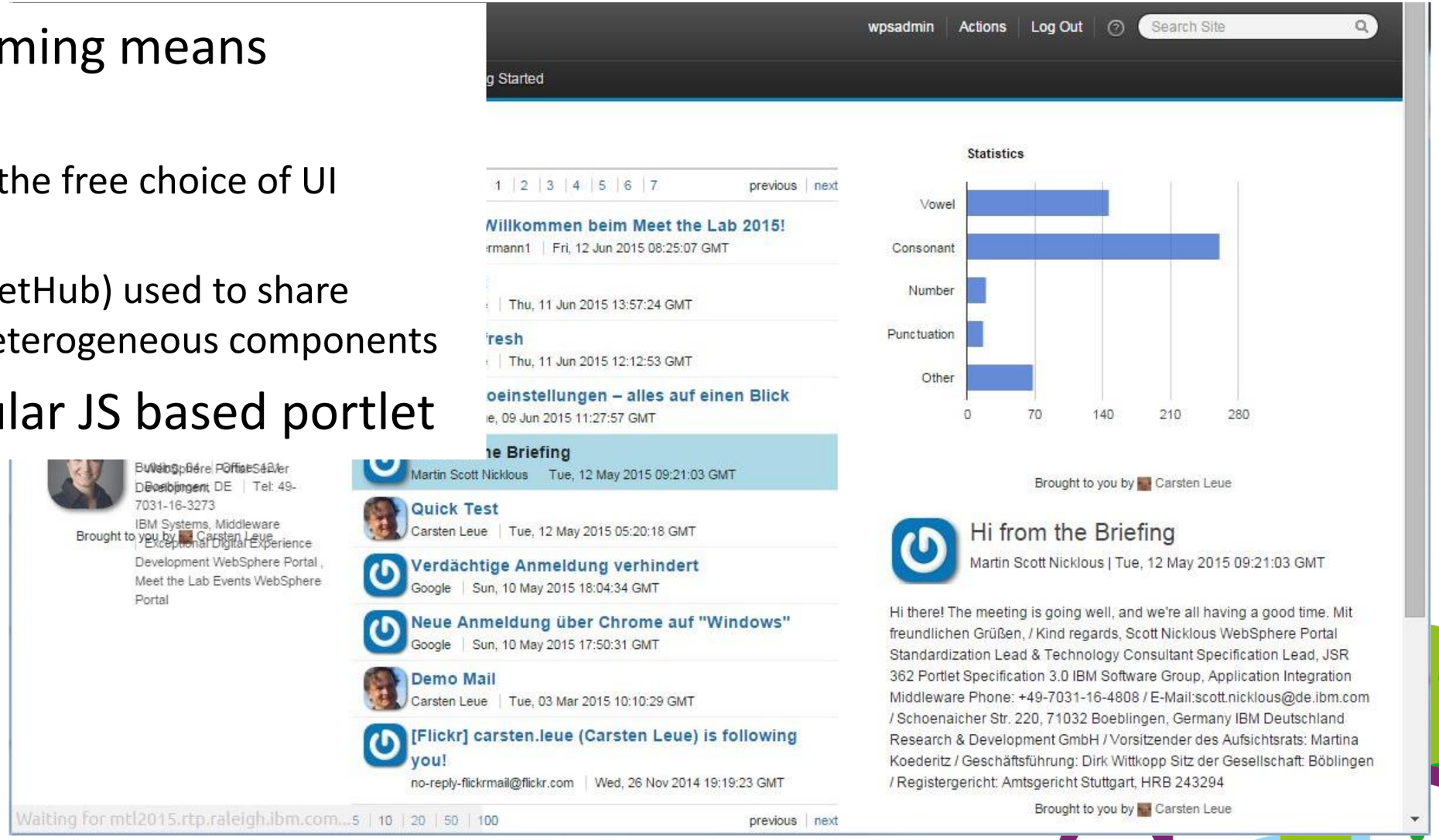
Portlet JSP

```
<%-- root element of the portlet --%>
<div id="${ns}_root">

<%-- anchor used API attached to the root element --%>
<a href="#" onclick="${ns}_root.increment()">inc</a>

</div>

<%-- bootstrap the library with the
     scope of the portlet--%>
<script type="javascript">
myAPIv1(${ns}_root);
</script>
```

Bootstrapping
For convenience make use of the **Named access on the Window object** feature of HTML

Playing with Friends – Angular JS

- Portlet Programming means flexibility
  - Each portlet has the free choice of UI framework
  - Portlet API (PortletHub) used to share context across heterogeneous components
- **Showcase**: Angular JS based portlet



Angular JS and Portlet Hub: Tips and Tricks

Portlet (JS)

Namespace
There might be more than one Angular application on the page, so use proper **namespacing**.

Bootstrap
Bootstrap the application as soon as the PortletHub is ready

```
/**
 * Callback invoked if the initialization of the
 * portlet hub is complete
 */
function xInit(hubAPI) {

/**
 * Assemble our angular application
 */
var xNamespace = aRoot.dataset.ns,
xNgApp = angular.module(xNamespace + "Module", []);

xNgApp.controller(xNamespace + "Controller", [ "$scope",
"$http", function(aScope, aHttpService) {
  // do something sensible
} ]);

/**
 * Bootstrap the angular portlet
 */
angular.bootstrap(aRoot, [ xNamespace + "Module" ]);
}
```

Scope
Limit the scope of the Angular application to the root node of the portlet

Angular JS and Portlet Hub: Tips and Tricks

Portlet Markup

Namespace
Controller has to be namespaced. From there on, we just use short aliases.

```
<%-- This is the main application part. The business logic is
encapsulated
in the controller. Note the use of the namespace to allow for
multiple instances of this portlet on the page. %>
<div ng-controller="${ns}Controller as list">

    <!-- list of folders -->
    <div ng-hide="list.showMails">

    <!-- iterate over each folder -->
    <tr ng-repeat="folder in list.items.div.default">

        <a href="#" ng-click="list.selectFolder(folder.href)">
        {{folder.name}} </a>

    </tr>

    </div>
</div>
```

Markup
The markup looks just like standard Angular JS markup. The only portlet specific part is the controller namespace.

Angular JS and Portlet Hub: Tips and Tricks

Synchronize State Updates

- Whenever the state of the portlet changes, update the JS state of the Angular JS model.
- Notify Angular about the changes via $apply()

Portlet Script

```
function xOnStateChange(aType, aState) {

    // decode the state
    var params = xState.parameters;

    // load some data
    aPortletInit.createResourceUrl({},
    _CACHE_LEVEL_PORTLET).then(aHttpService.get).then(function(xhr)
    {
        // update the mails
        xThis.items = xhr.data.mails;

        // notify angular about these changes
        aScope.$apply();
    });

}
```

Data Updates

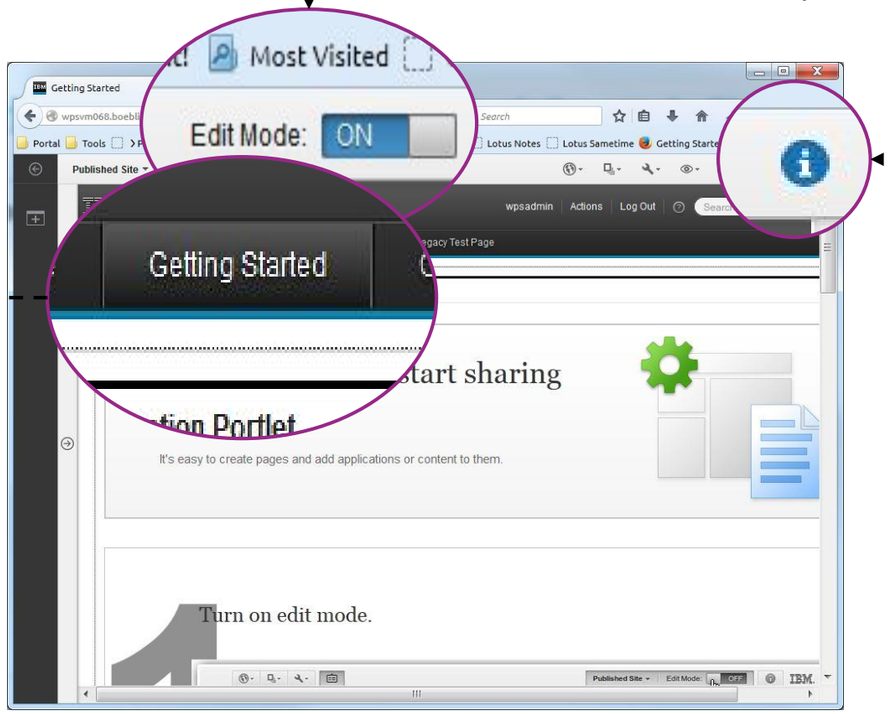
- Fetch updates to the data via serveResource
- Convenient access to XHR via $http service of AngularJS.

Interaction with System Level Context

- WebSphere Portal manages system level state
  - Page Selection, Label expansions, …
  - Edit Mode, Info Mode, …
  - Locale
- Traditionally this state is manipulated from Theme Components
  - E.g. via urlGeneration tag
  - Dynamic Spot JSPs …
- **New in 8.5:** System level state is also represented as Public Render Parameters
  - Portlets can subscribe the system state by declaring public parameters in their deployment descriptor
  - State can be both read and written
  - Support for Client Side interactions and WSRP

Edit-Mode
On/Off

Info-Mode
On/Off



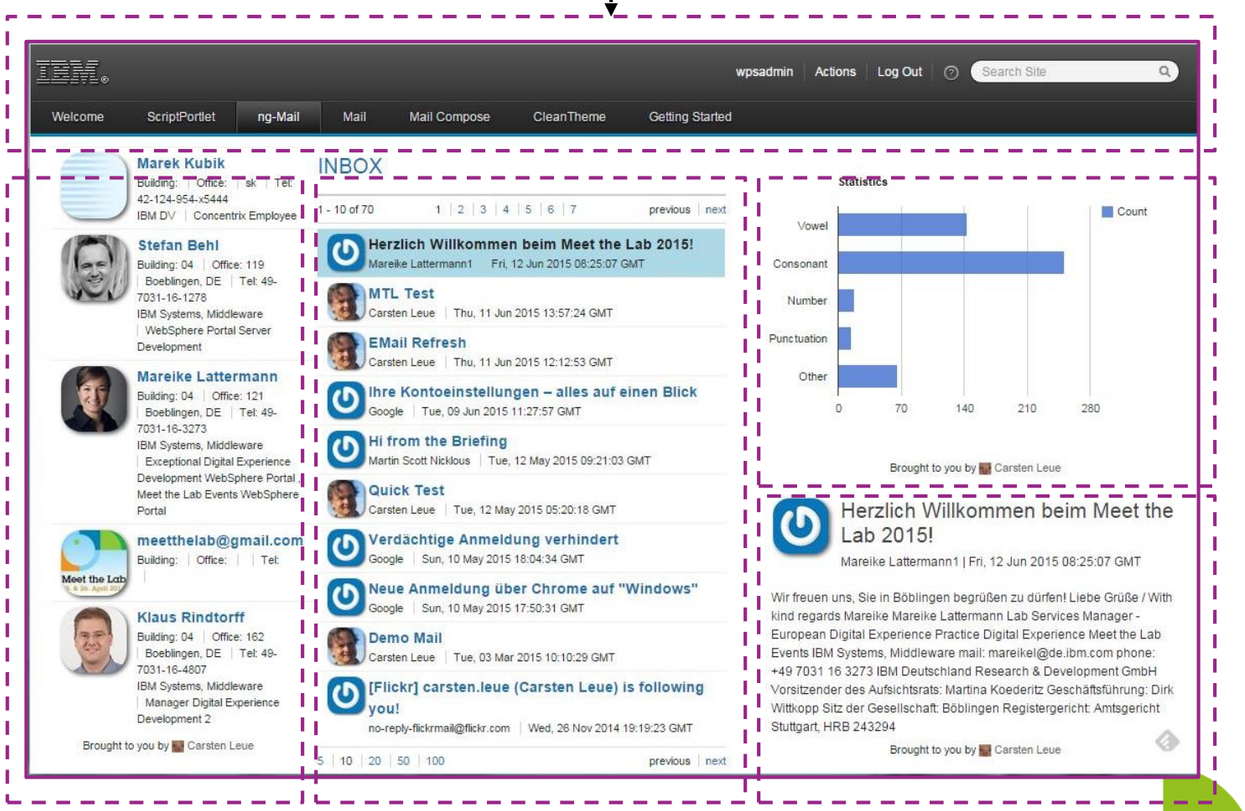
Selection
ID of the currently selected portal page.

Locale
Locale override of the display locale.

Use case - *Themeless* Pages

- Represent traditional theme level components as portlets
  - E.g. page navigation portlet
- The theme does not contribute markup
- Layout templates contain prefilled content spots
- Advantages:
  - All components on a page share the same programming model: **Portlets**
  - Each component on the page can be coded independently as local, remote or client side
- Disadvantage:
  - Each page has its own instance of the theme level portlet

Navigation
The navigation portlet realizes the page transitions via public render parameters



Portlets
Each aspect of the page is represented by a portlet.

Script Portlet and Portlet Hub

- The IBM Script Portlet
  - Enables a script developer to create portlets for IBM WebSphere Portal with JavaScript, CSS, and HTML.
  - Content stored in WCM, no J2EE deployment required
- Synergy with the Portlet Hub
  - Script Portlet can make use of the Portlet Hub to work with private and public render parameters
  - React to updates of public render parameters
- Limitations
  - Fixed set of public render parameters as defined by the WCM Rendering Portlet
  - No access to “serveResource”, yet. Can work with globally defined REST services, though

Namespacing
Use a WCM rendering plugin to express the namespace.

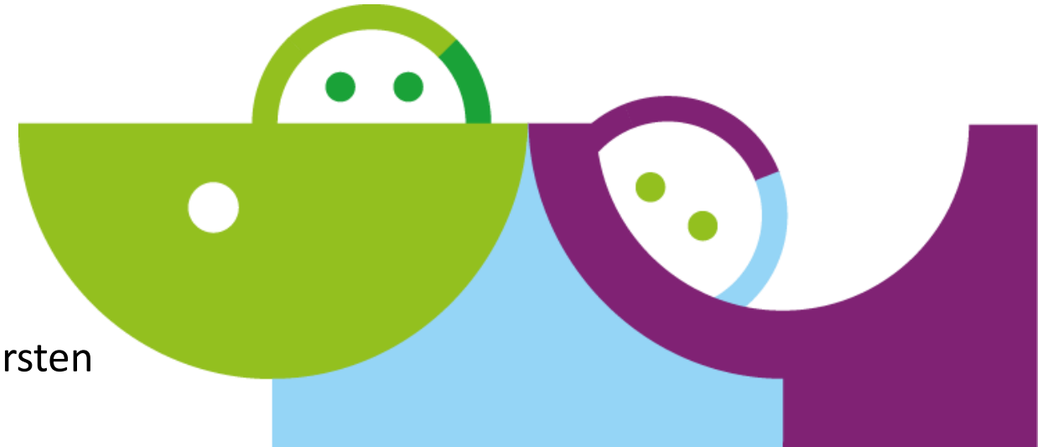
```
<div class="mail-faces-script lotusui30 hidden"
id="[Plugin:ScriptPortletNamespace]root"
data-ns="[Plugin:ScriptPortletNamespace]"
data-content-
handler="[Plugin:ScriptPortletNamespace]ContentHandler">
…
</div>
```

```
<!-- base URL for the contenthandler -->
<a
  href='[Plugin:RenderURL copyCurrentParams="false"
         uriMode="download" uri="" escape="xml"]'
  style="display: none“
  id="[Plugin:ScriptPortletNamespace]ContentHandler">
</a>
```

REST Services
Access to the base URL to REST services in WebSphere Portal (Resolver Framework). Notice **`uri=""`**

Programming Model Patterns

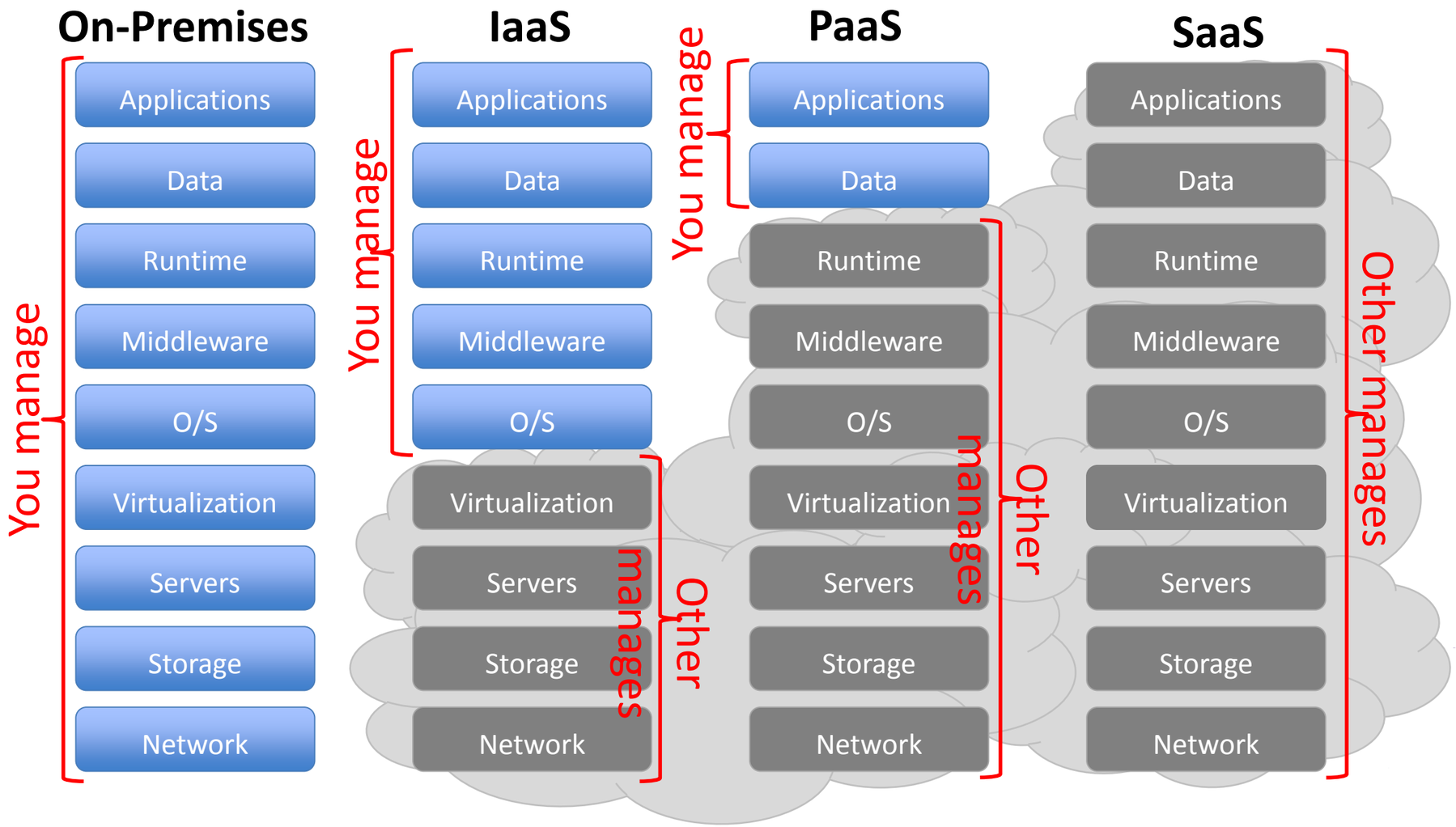
Focus on Cloud Patterns



June 17th 2015

Client Side Programming – Dr. Carsten Leue

Abstraction Level



Cloud Infrastructure

Abstraction Level (simplicity)

- IaaS: abstraction level = server/storage (ready to install)
- PaaS: abstraction level = OS platform/middleware (ready to build)
- SaaS: abstraction level = application (ready to consume)

Cloud Topology (complexity)

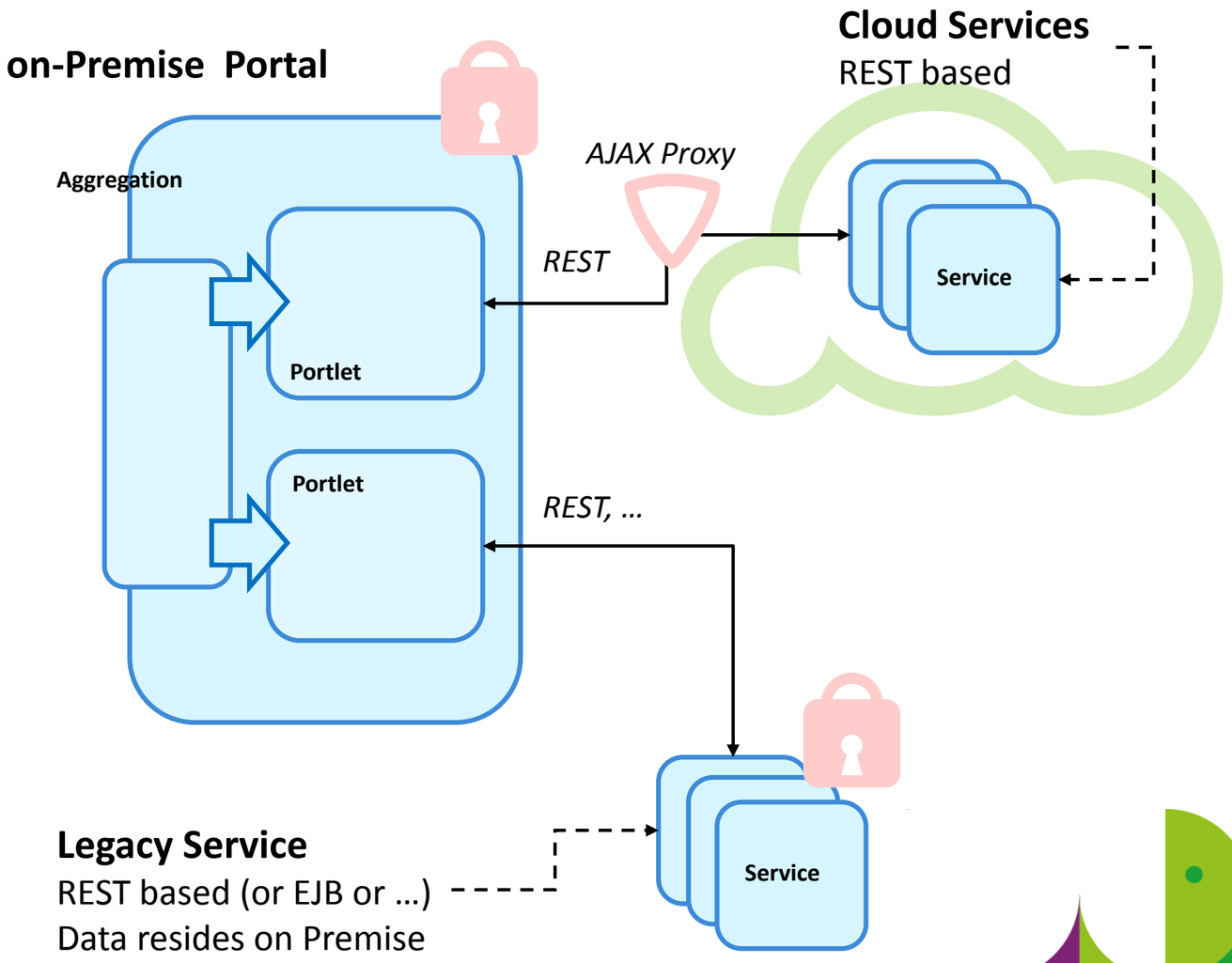
- Private Cloud: Operated by and restricted to a single enterprise.
  - Isolation: use of dedicated resources
- Public Cloud: Operated by a service provider who grants access to a large audience of unrelated enterprises
  - Isolation: shared resources
- Hybrid Cloud: Mix of private and public cloud as well on-premises services (in any kind of abstraction level)
  - Let‘s face it: This sounds like the most realistic situation for most of us

Application Development in the Cloud

- Composed of Services
  - Data based backend services
  - Services come together on the Glass or on the Server
- Developed in multiple Programming Languages
  - Web: JavaScript, Ruby, Java, PHP
  - Mobile: iOS, Android, SDK
- Integrate with existing systems
  - Data is often located in multiple places (public, private, traditional data center)
  - Existing systems may not scale at the same the level of cloud applications
- Access to administrative tasks is restricted and limited

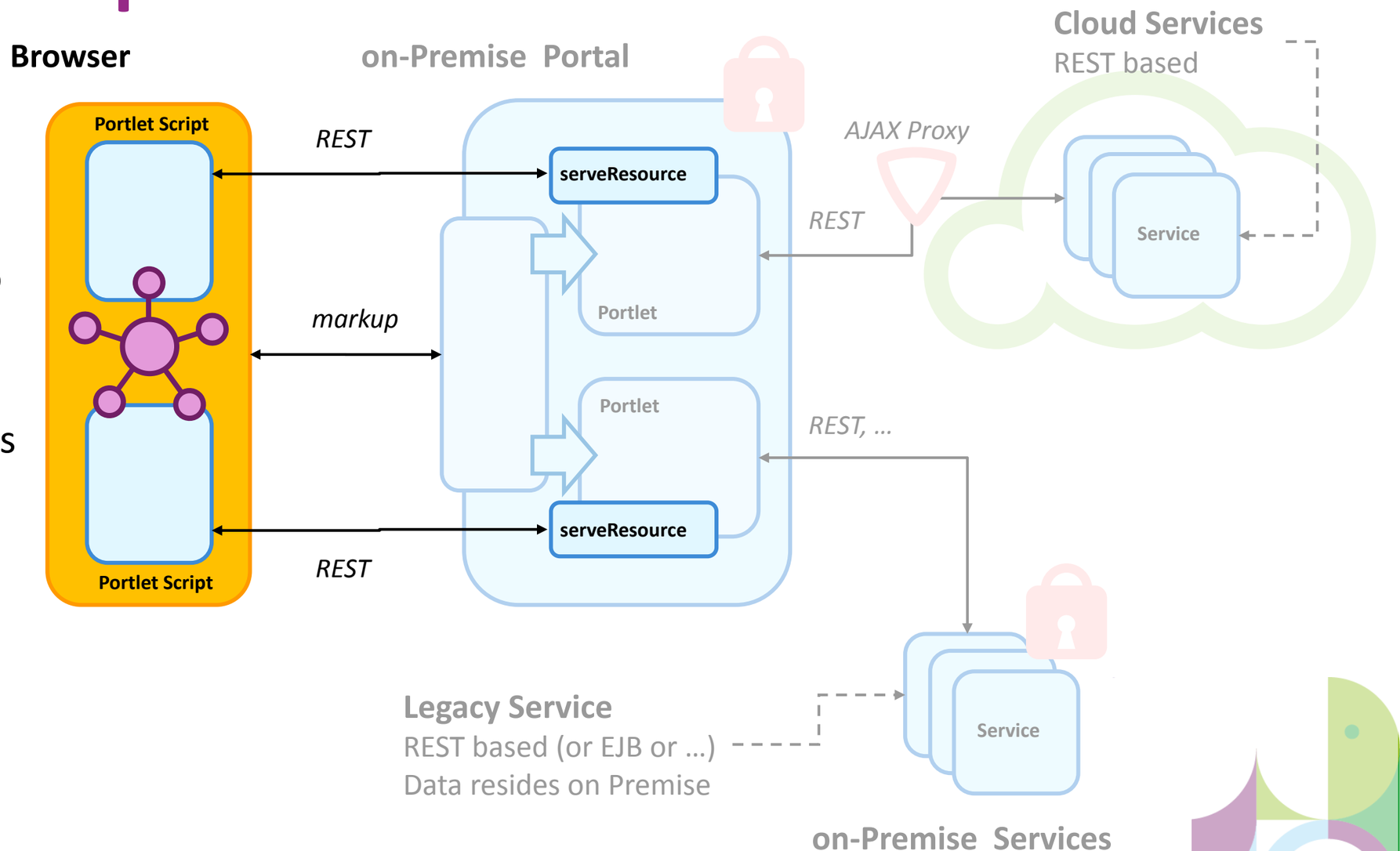
Data Consumption in an on-Premise Portal

- Traditional Approach
  - Implement UI components as Portlets
  - All data is fetched by the server side portlets logic from backend services
  - Server controls access to services that run on Premise or in the Cloud



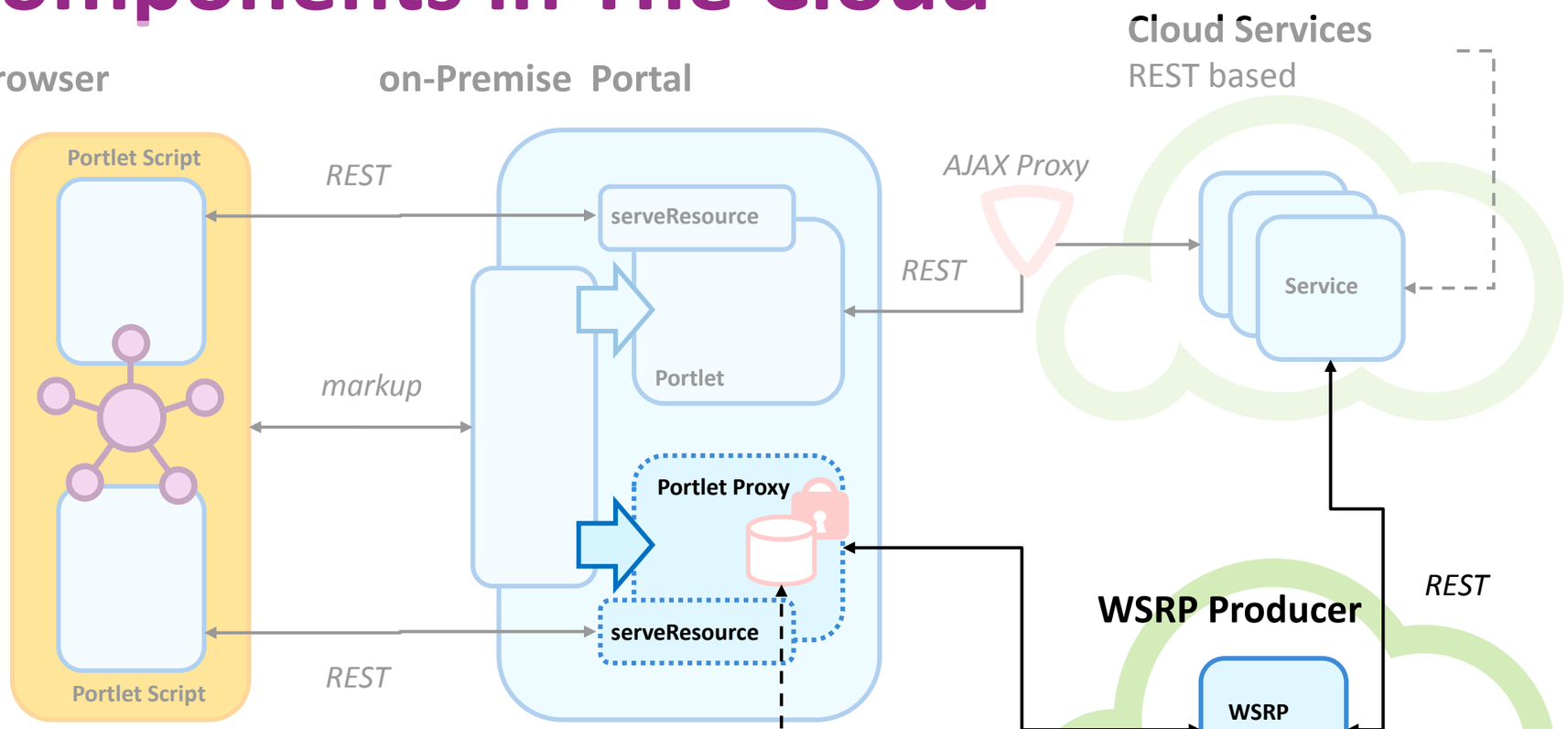
Data Consumption in an on-Premise Portal

- Client Centric Approach
  - Portlet Hub based UI, e.g. on top of Bootstrap and Angular JS
  - Data access via serveResource, acting as a proxy



UI Components in The Cloud

- Consume markup services from the cloud
  - On Premise portal used as an aggregation platform and for access control
  - Business logic completely offloaded to the cloud via WSRP
- Data security
  - Data in portlet preferences is persisted on the consumer (however sent to the cloud)
  - Extra data may be stored in the cloud by the remote portlet

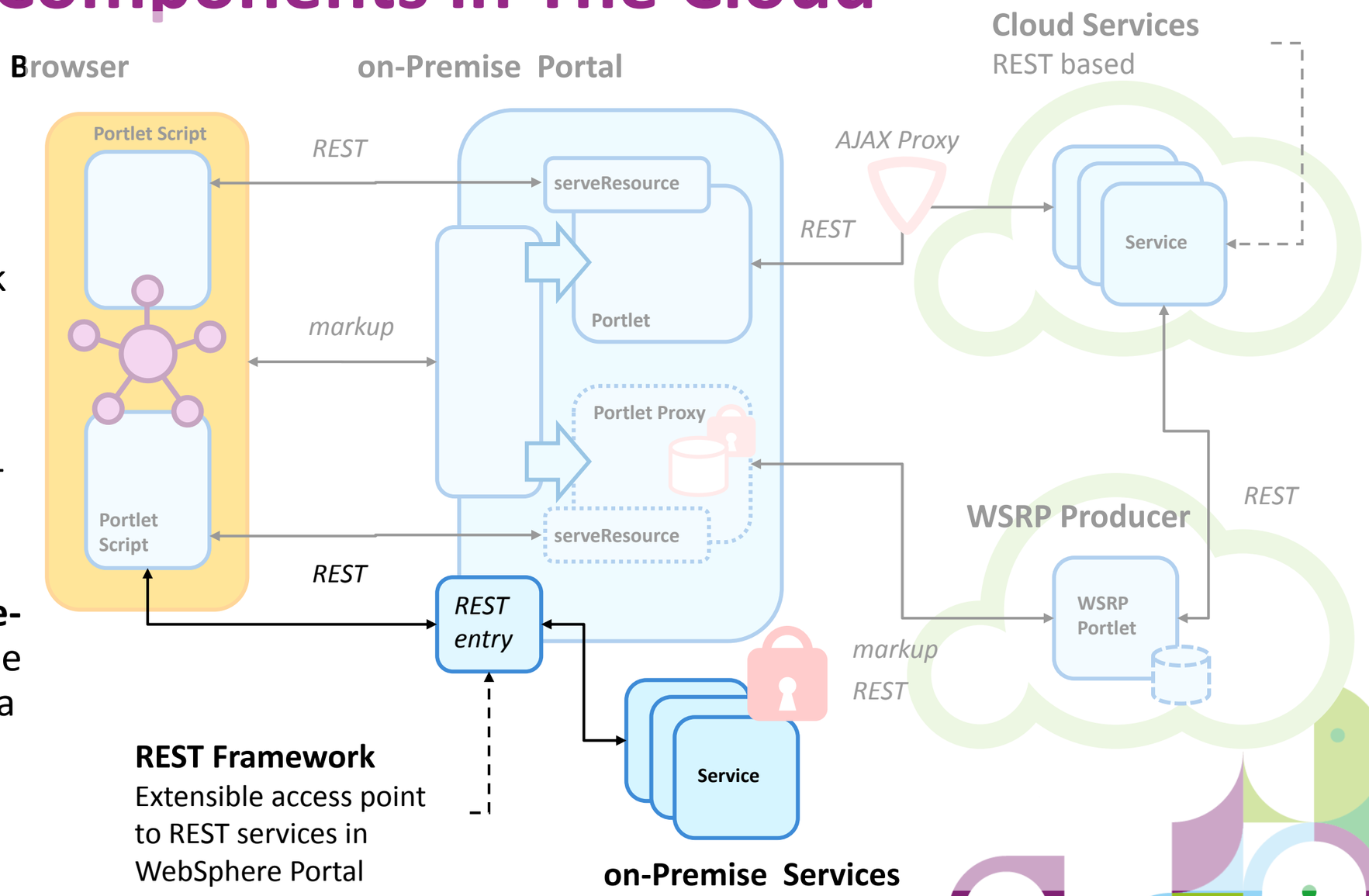


Portlet Preferences
Preferences are stored on the consumer side

Extra Data
Optionally the remote portlet can store extra data in the cloud

UI Components in The Cloud

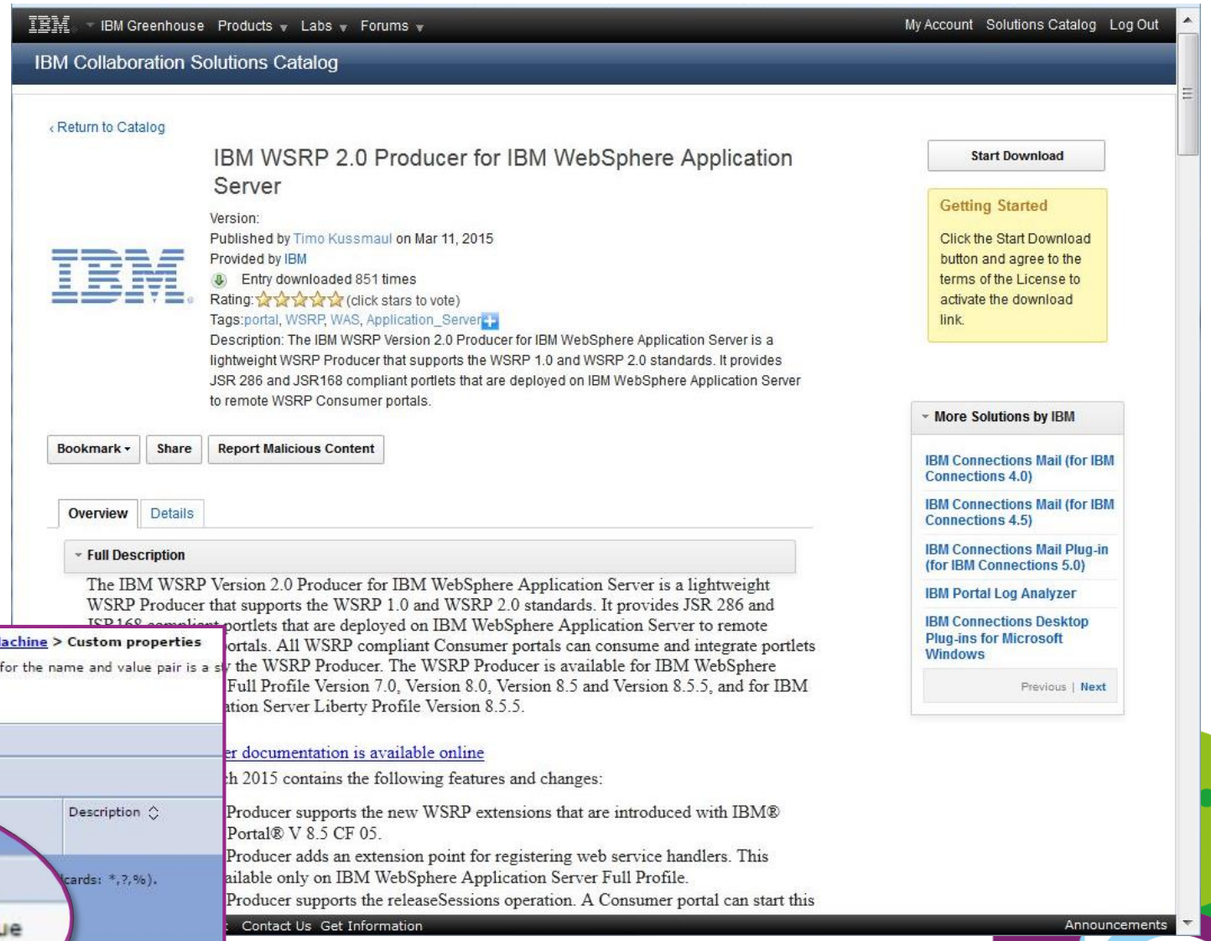
- Remote Portlets consume on-Premise data on the glass
  - REST service framework provides access to on-Premise services
  - Script accesses these services to mash-up on-Premise data into the remote markup
  - No problems with **Same-Origin-Policy** as both the script as well as the data are served by the on-Premise Portal

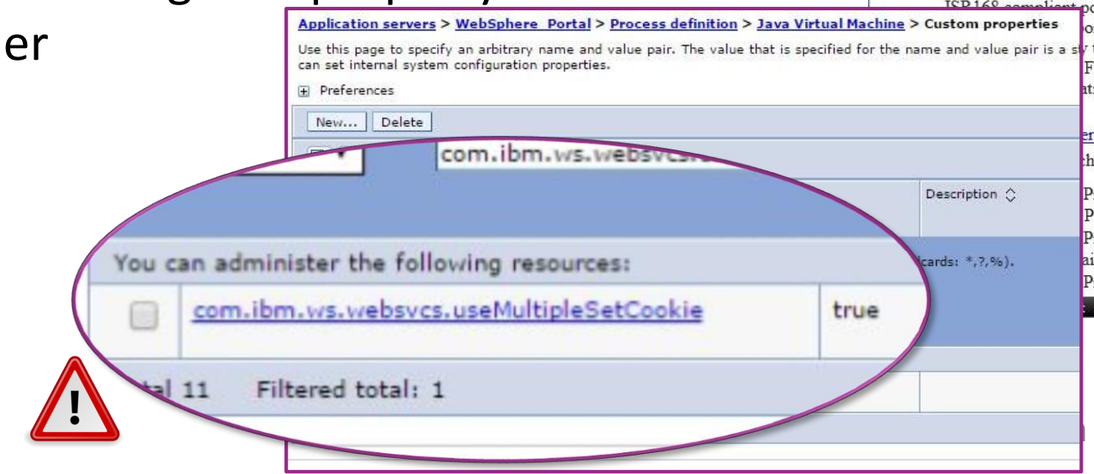


WSRP Services in The Cloud

- WSRP Producer on Liberty
  - Full support of the JSR 286 compliant portlets on WAS Liberty
  - Fast development cycle thanks to the ultra slim Liberty runtime
- Support for Bluemix
  - Since March 2015 support for the Liberty Profile on Bluemix
  - Set the following JVM property on the WSRP Consumer

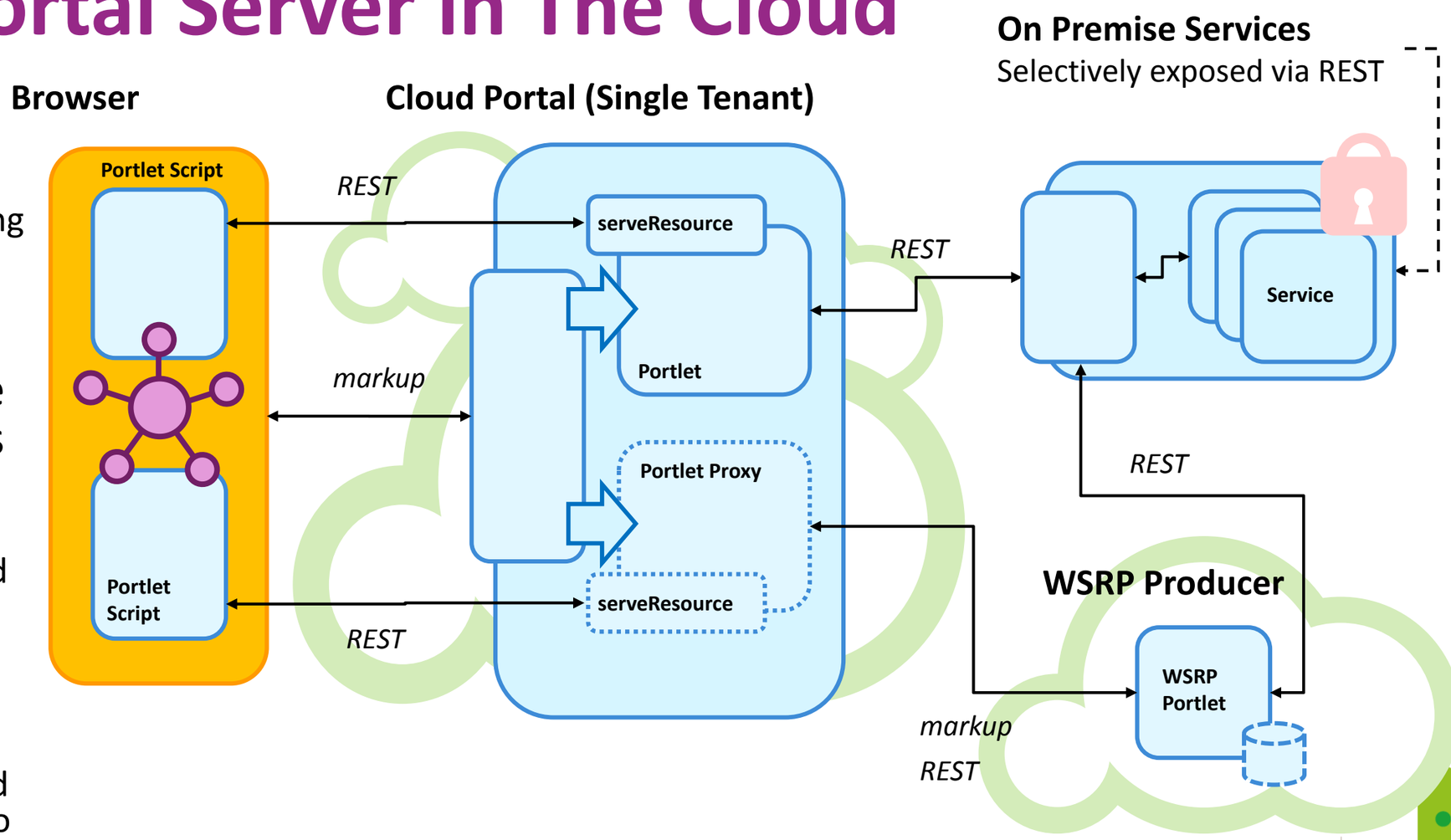
Available on the **IBM Solutions Catalogue**





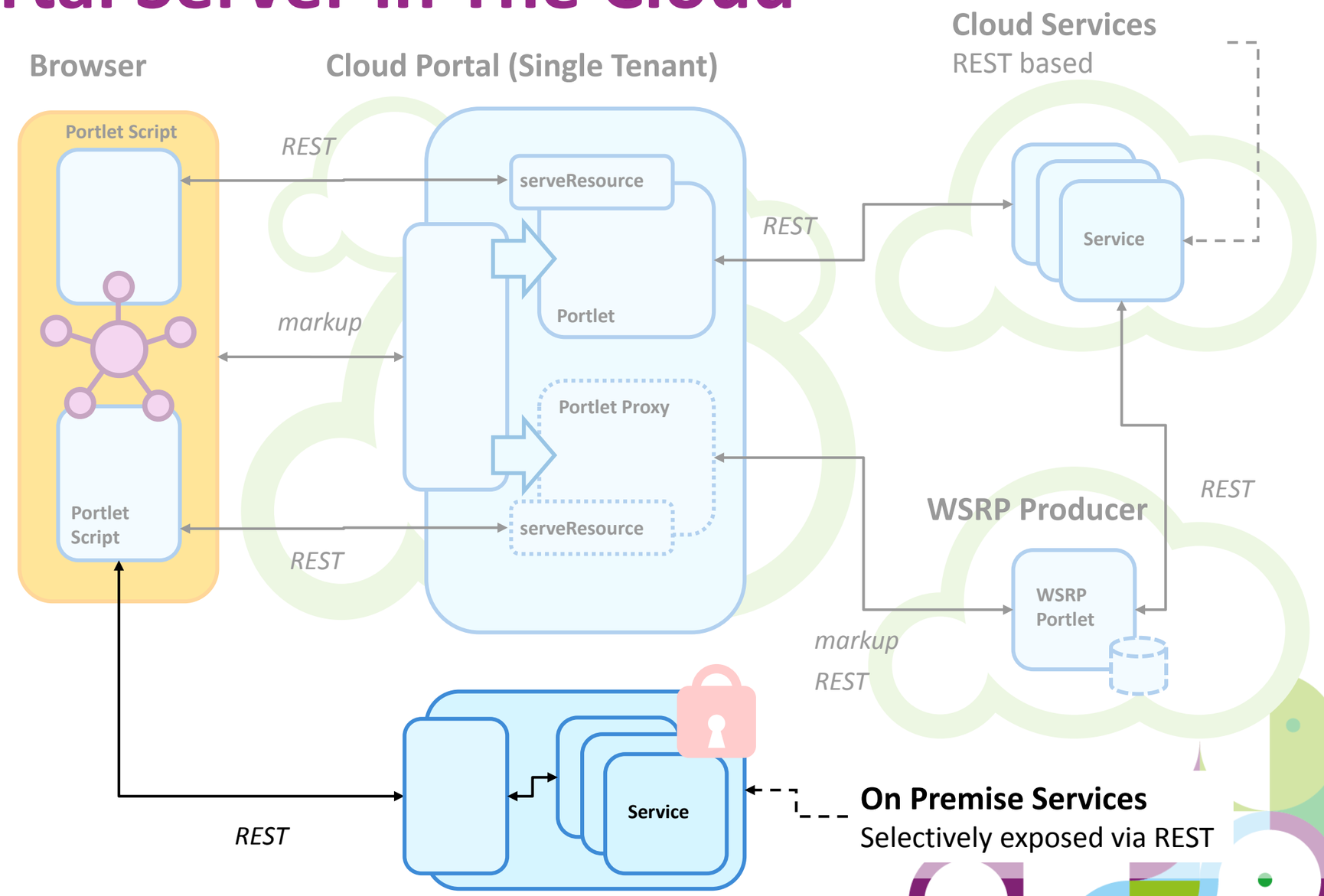
Portal Server in The Cloud

- Portal runs in the Cloud (e.g. DX on Cloud)
  - Single tenant as PaaS Offering
  - Data stored in the cloud, however managed by the Customer
- Integration with sensitive data via backend services
  - Services are exposed via a REST interface
  - Portlets running in the Cloud consume the services and render data
- Challenges
  - On Premise services have to be accessible from the Cloud (probably need Gateway into the Intranet)

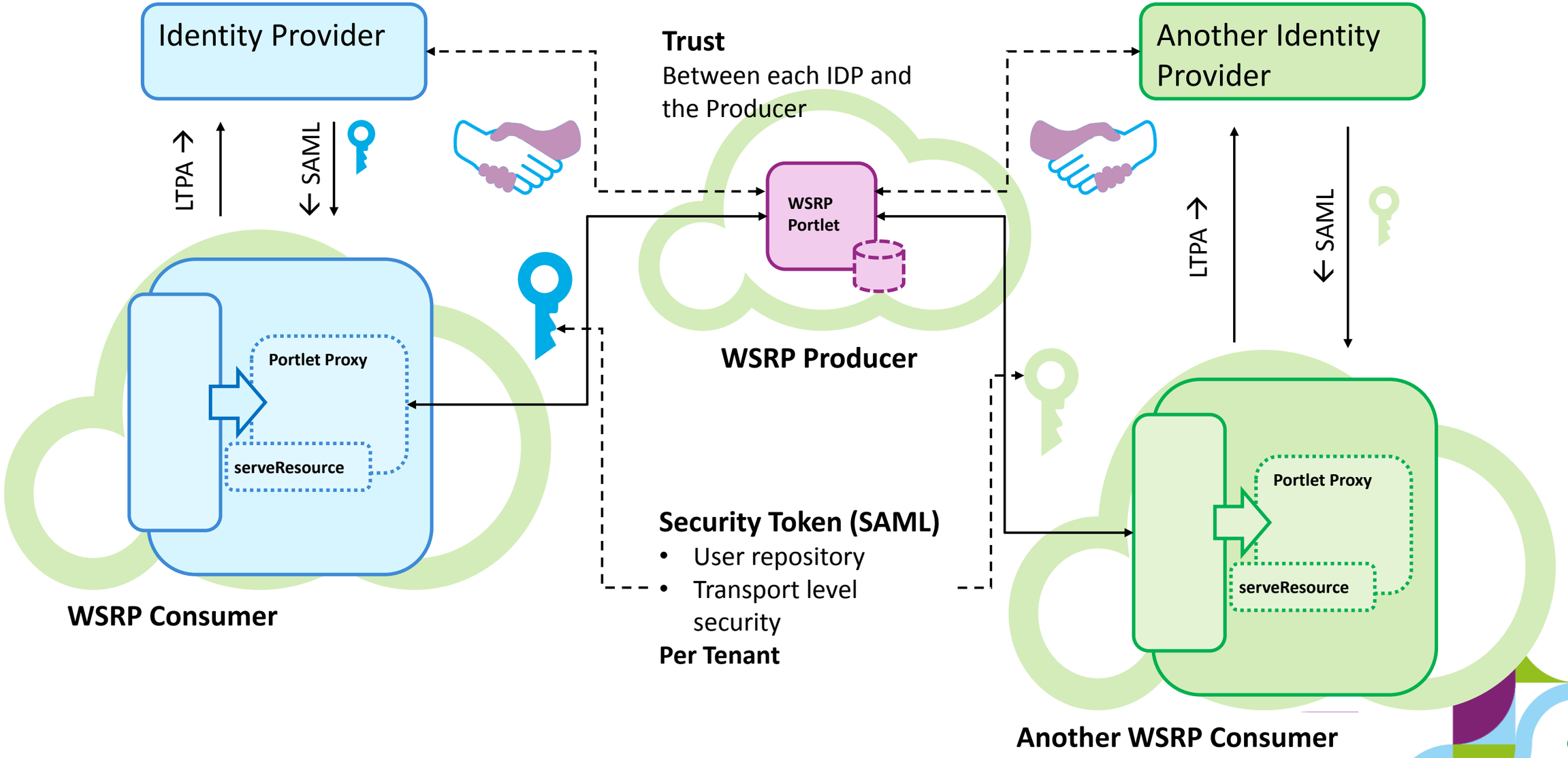


Portal Server in The Cloud

- Portal runs in the Cloud (e.g. DX on Cloud)
  - Single tenant as PaaS Offering
  - Data stored in the cloud, however managed by the Customer
- Integration with sensitive data on the Glass
  - Javascript calls to REST services that are hosted on-Premise
  - Data is not accessed directly by code running in the cloud
  - No risk of caching sensitive date in the cloud
- Challenges
  - Single-Sign-On
  - Same-Origin-Policy
- Advantages
  - If the browser runs in the Intranet, no public Gateway to the services required



Multi Tenancy and WSRP



Summary

- The approach to build a Web Application from coordinated components is an important use case
  - Both for modern client side and cloud based applications
- The Portlet Programming Model provides a suitable abstraction level for this kind of applications
- JSR 362 makes the Portlet Programming Model ready for rich client side applications

Ask for the Portlet Hub on your **Evaluation Sheets**, if you are interested!

Questions