

Event Handling in AEM

Agenda:

- Describe Sling scheduler
- Create a Job consumer
- Discuss event modeling
- Create a resource listener



Event Handling in AEM

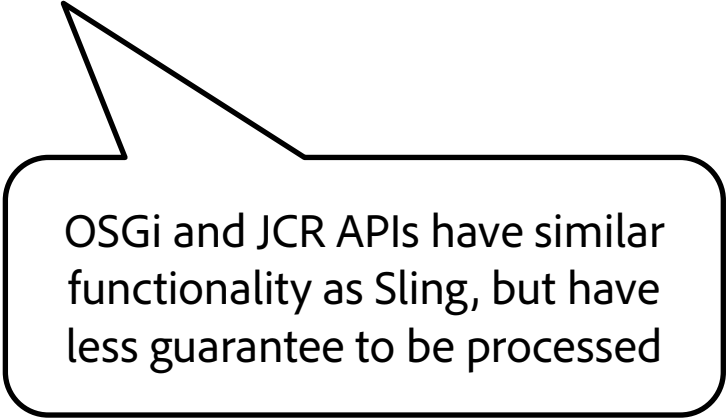
Event handling can be done in virtually any framework in AEM

- OSGi, with event handlers
- JCR, with observation
- Sling, with Sling jobs
- AEM application, with workflows and launchers

Choose based on need

Use higher level APIs for events

- Sling API
 - High processing and high efficiency
 - All code and No management UI
 - Guaranteed to be handled
 - Completely automated after deployed
- AEM Workflows
 - Tightly integrated with AEM applications
 - Management UI
 - Historical data
 - Used primarily for business cases that need human interaction



OSGi and JCR APIs have similar functionality as Sling, but have less guarantee to be processed

Processing events on a Schedule

Sling Jobs Scheduler

- Enables you to easily schedule jobs for your application
- Scheduled Jobs can be executed
 - At a specific time
 - Regularly at a given period
 - At times given by a cron expression
- Primary goal is to load a Sling Jobs queue with work
 - Sling Jobs do the actual processing
- Multi-scheduled jobs can be created with Factory OSGi configurations

More on Sling Jobs Later...

Scheduling Jobs

Retrieve the Job Manager

```
70  @Reference
71  private JobManager jobManager;
```

Setup the Job

```
99  jobBuilder = jobManager.createJob(StockImportSchedule
100  // Create a properties map that contains the configur
101  HashMap<String, Object> jobProps = new HashMap<>();
102  jobProps.put(JOB_PROP_SYMBOL, config.symbol());
103  jobProps.put(JOB_PROP_URL, config.stock_url());
104
105  jobBuilder.properties(jobProps);
106  scheduleBuilder = jobBuilder.schedule();
```

Schedule it

```
107  scheduleBuilder.cron(config.cronExpression());
108  theScheduledJob = scheduleBuilder.add();
```

Schedules can be set based on

- Cron expressions
- Specific Dates
- daily, hourly, monthly, weekly, and yearly

Use Case | Import Stocks

ACME wants to use 3rd party data sources on their website. They don't want to perform external calls at runtime since it may impact page load time. Since the data only needs to be *near real time*, the data could be stored within AEM so it can be cached on the Dispatcher and CDN. They also want to empower their business users to be able to create their own data sources as needed.

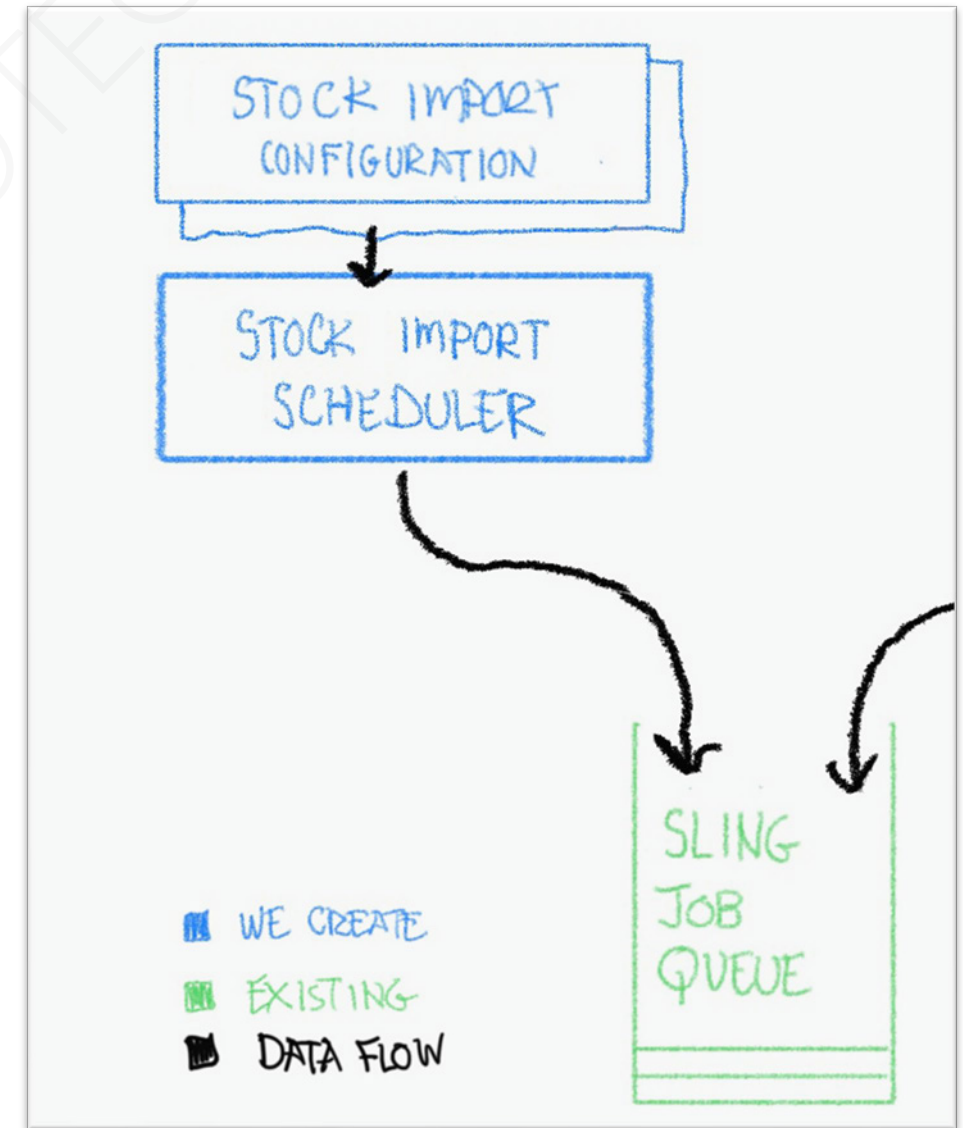
The 3rd party data ACME wants to display is stock data. Business users need the ability to import *near real time* stock information into AEM to use on their website, without developer intervention.

Throughout this module you will build background events to pull in the 3rd party data based on ACME's requirements

Import Stocks | Job Scheduler

Schedule as many stocks as needed

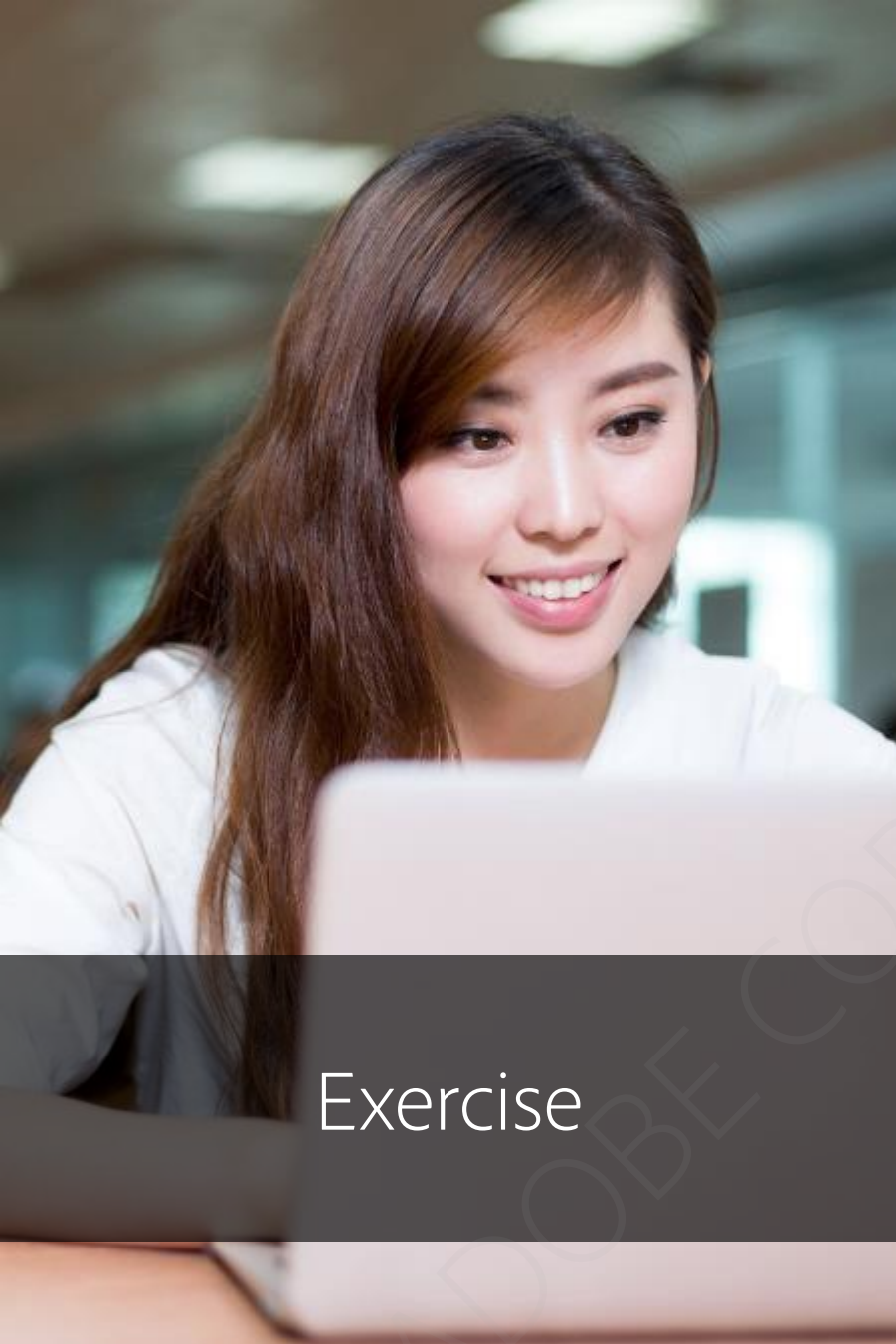
- OSGi configuration factory can create many importers
 - Developers can create initial json config files for stocks
- Each configuration creates a scheduler
- Scheduler creates Sling Jobs
- Sling Jobs are added to the Sling Job queue



Actual job processing (import 3rd party data) will occur in the next exercise

Creating Scheduled Jobs

```
01. public class StockImportScheduler {
02.
03.     @Reference
04.     private JobManager jobManager;
05.
06.     @Activate
07.     protected void activate(Configuration config) {
08.         startScheduledJob(config);
09.     }
10.     @Deactivate
11.     protected void deactivate(Configuration config) {
12.         removeScheduler(config);
13.     }
14.
15.     private void startScheduledJob(Configuration config){
16.
17.         jobBuilder = jobManager.createJob("com/adobe/training/core/jobs/stockimportjob");
18.
19.         scheduleBuilder = jobBuilder.schedule();
20.         scheduleBuilder.cron("0 0/2 * * * ?");
21.         theScheduledJob = scheduleBuilder.add();
22.     }
23.
24.     private void removeScheduler(Configuration config) {
25.         logger.info("*****Removing '{}' ScheduledJob, with ID: '{}'", config.symbol(), schedulerID);
26.         theScheduledJob.unschedule();
27.     }
28. }
```

Exercise 1: Schedule a Sling Job

In this exercise, you will create an OSGi component that creates a new scheduled Sling Job based on the OSGi configs provided. This OSGi component is also a config factory implying there can be many instances. Note that this component only adds scheduled jobs to the queue and does not actually process the Jobs.

- Task 1: Create a Sling Scheduler
- Task 2: Create JSON configuration files
- Task 3: Install via command line
- Task 4: Verify configurations

Sling Job Events

A job event is

- Always processed
- Handled only once
- Synchronous per Job queue

Job Topics

- Unique to a job and queue
- Example: `com/adobe/training/core/jobs/stockimportjob`
- Allows for same jobs to be managed and maintained by the same queue

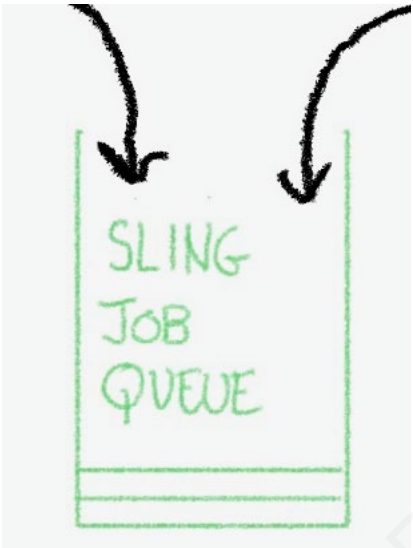
To process a job event from a queue, the service must implement:

- `org.apache.sling.event.jobs.consumer.JobConsumer`

Job Consumer Pattern

- Jobs are created and identified by a topic
- Jobs are added to the queue by schedulers, listeners, or other components
- Jobs are processed by Job Consumers

```
97 private void startScheduledJob(StockImportConfiguration config){
98     jobBuilder = jobManager.createJob("com/adobe/training/core/jobs/stockimportjob");
```



```
64 @Component(
65     immediate = true,
66     service = JobConsumer.class,
67     property = {
68         JobConsumer.PROPERTY_TOPICS += "com/adobe/training/core/jobs/stockimportjob"
69     }
70 )
71 public class StockDataWriterJob implements JobConsumer {
72     @Override
73     public JobResult process(Job job) {
74         // Do something to process the job
75     }
```

Sling Job Console

Console for all job topics, queues, and statistics

Can be accessed locally for development:

- Web Console > Sling > Jobs

Adobe Experience Manager Web Console

Jobs

[Main](#) [OSGi](#) [Sling](#) [Status](#) [Web Console](#) [Log out](#)

Apache Sling Job Handling

Apache Sling Job Handling: Overall Statistics

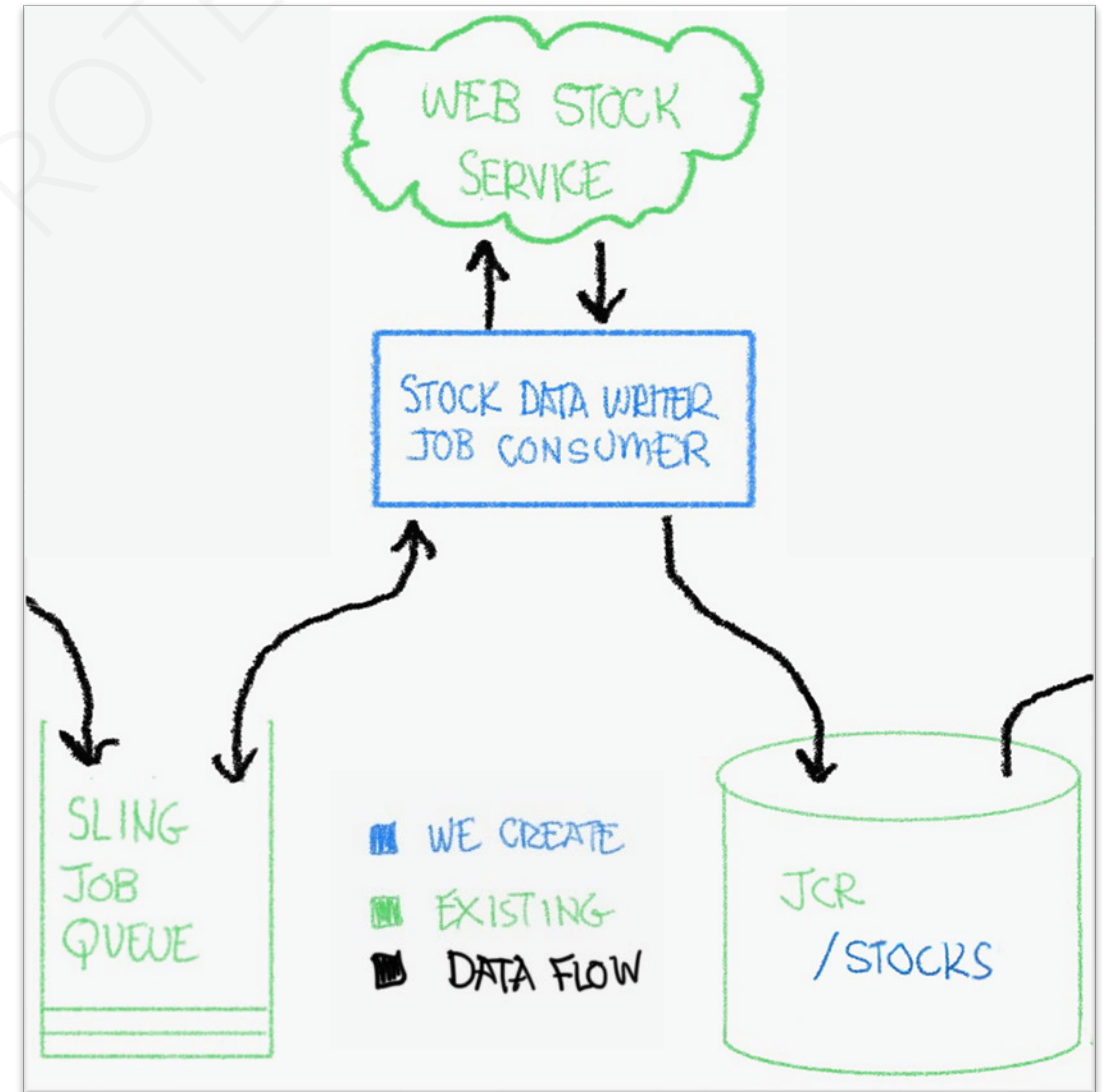
Reset Stats

Start Time	10:13:49:212 2017-Apr-13
Local topic consumers:	<div>ADD-ASSET-USAGE</div> <div>com/adobe/aem/formsndocuments/scheduler/formreplication</div> <div>com/adobe/cq/dam/assetmove</div> <div>com/adobe/cq/dam/dmassetreplicateonmodify</div> <div>com/adobe/cq/wcm/launches/autopromote</div> <div>com/adobe/cq/workflow/payload/move/job</div> <div>com/adobe/granite/maintenance/job/AuditLogMaintenanceTask</div> <div>com/adobe/granite/maintenance/job/DataStoreGarbageCollectionTask</div> <div>com/adobe/granite/maintenance/job/RevisionCleanupTask</div>

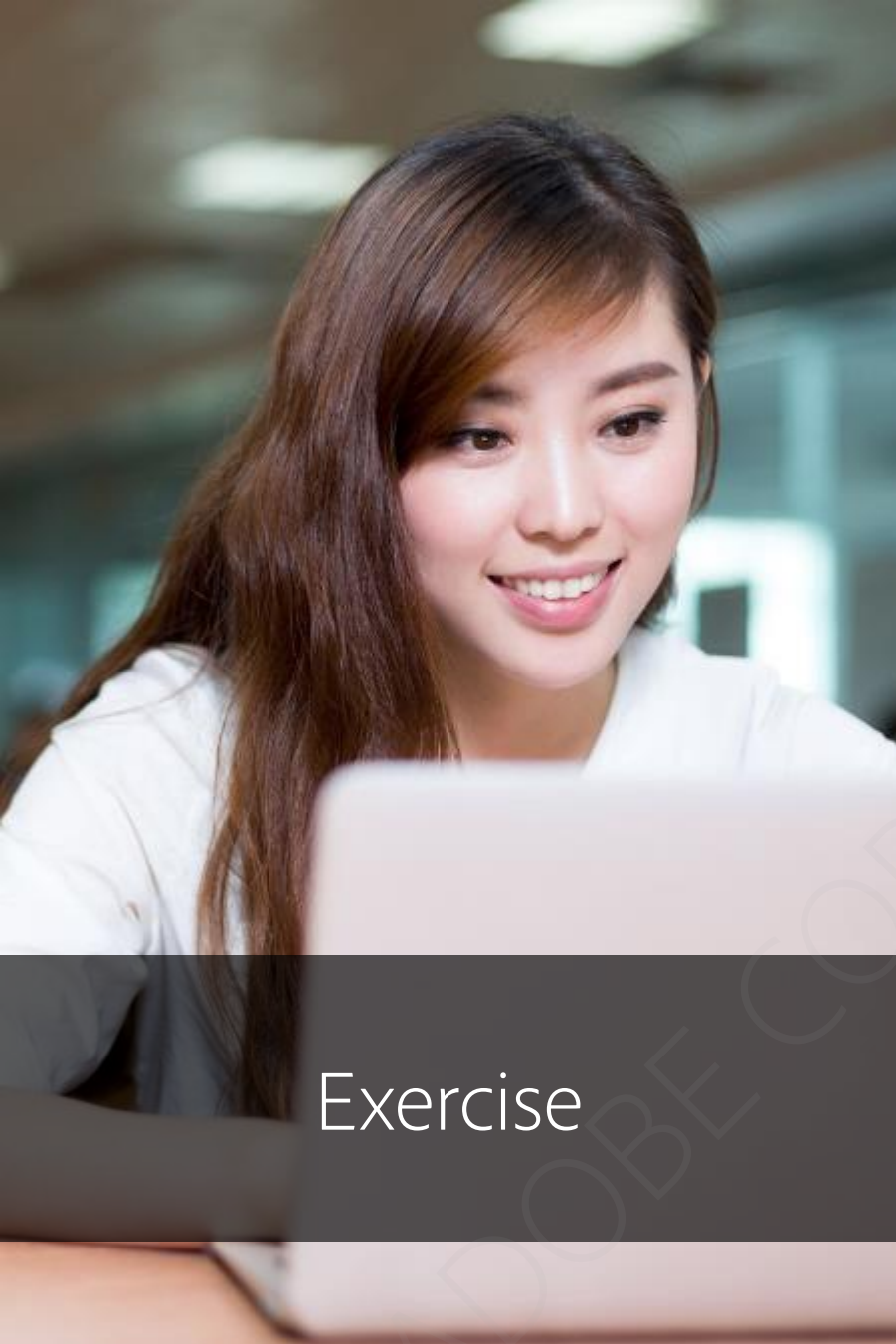
Import Stocks | Job Consumer

Reach out to the source and import desired info

- Stock symbol is contained in job metadata
- Use HTTP to get a json response for the stock info
- Save the desired stock info into the JCR



Business users will decide stock symbols in the next exercise



Exercise

Exercise 2: Consume Sling Job

In the previous exercise, you added a scheduled Sling Job with the JobManager. In this exercise, you will process the Sling Job that was put in the Job Queue. The Job consumer you will build imports data from an external API endpoint and then writes it into the JCR. For this example, we are using a dummy stock data endpoint located in git, though in realistic scenarios this method could be used for integrations and connecting other 3rd party sources to AEM.

- Task 1: Create a Sling Job Consumer
- Task 2: Install via command line
- Task 3: Test the service

Resource Listening Events

Mechanism that listens for resources changing within AEM

- Enables the application to receive notification changed and act on them
- Available via the Sling ResourceChangeListener interface

Listening Type Triggers:

- ADDED, CHANGED, REMOVED, PROVIDER_ADDED, PROVIDER_REMOVED

When an event occurs that falls within the change types:

- Sling calls the onChange() method with a changes object

The Resource Change Object

Each change generated by Sling is represented by a ResourceChange object.

This Object contains

- Change path – retrieved through String change.getPath()
- User who made the change– String change.getUserId()
- Get the Type– getType()
- Properties that caused the change via getter methods

A Change

- records the identity of the session that caused it.
- may contain arbitrary string data specific to the session that caused the change.
- records the time of the change that caused it.

ResourceChangeListener

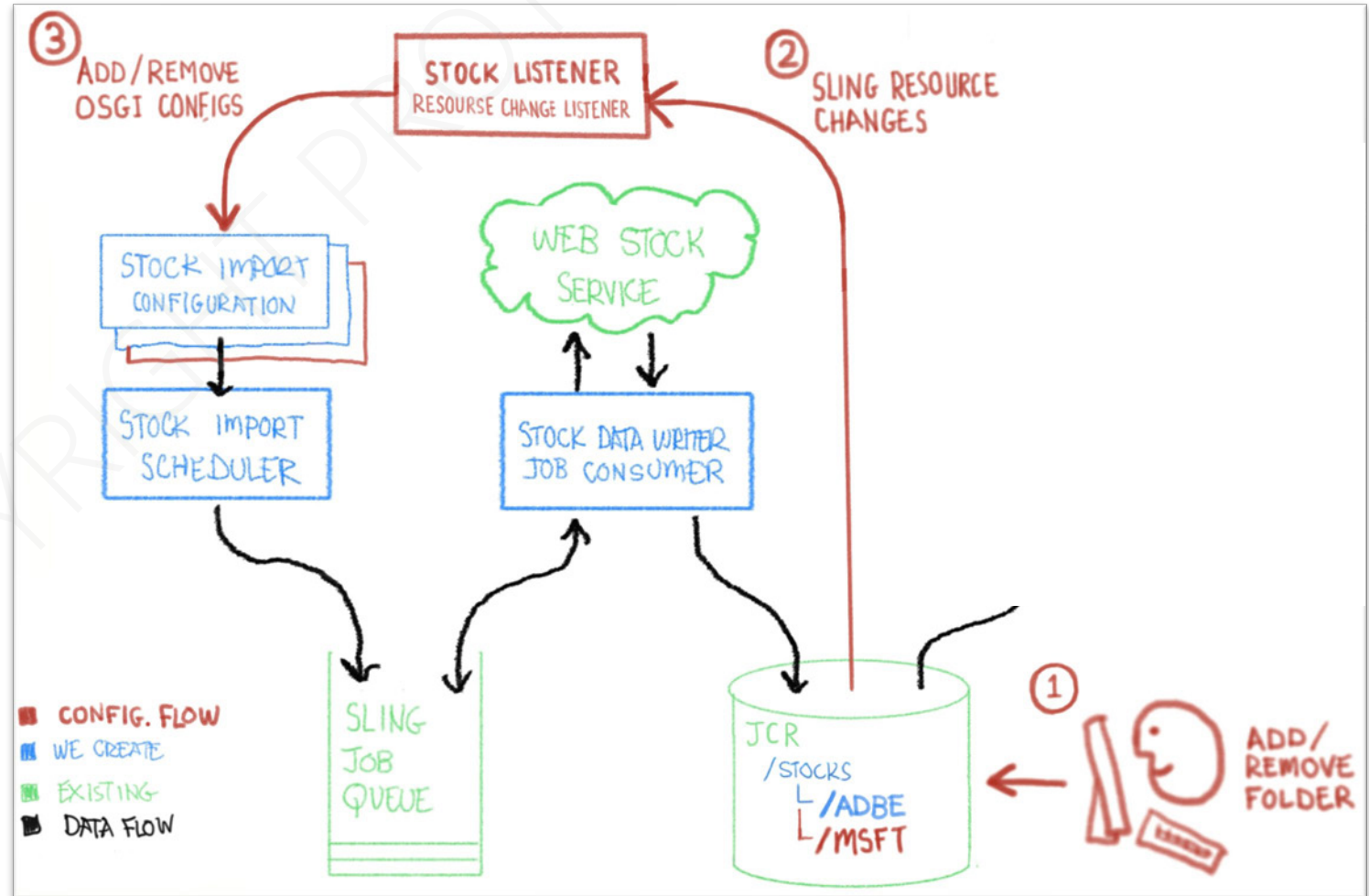
```
import org.apache.sling.api.resource.observation.ResourceChange;
import org.apache.sling.api.resource.observation.ResourceChangeListener;
@Component (immediate = true,
service = ResourceChangeListener.class
    property = {"resource.paths=/content/mypath ",
                "resource.change.types=CHANGED"})

public class MyResourceListener implements ResourceChangeListener {
    @override
    public void onChange(List<ResourceChange> changes) {
        // do something
    }
}
```

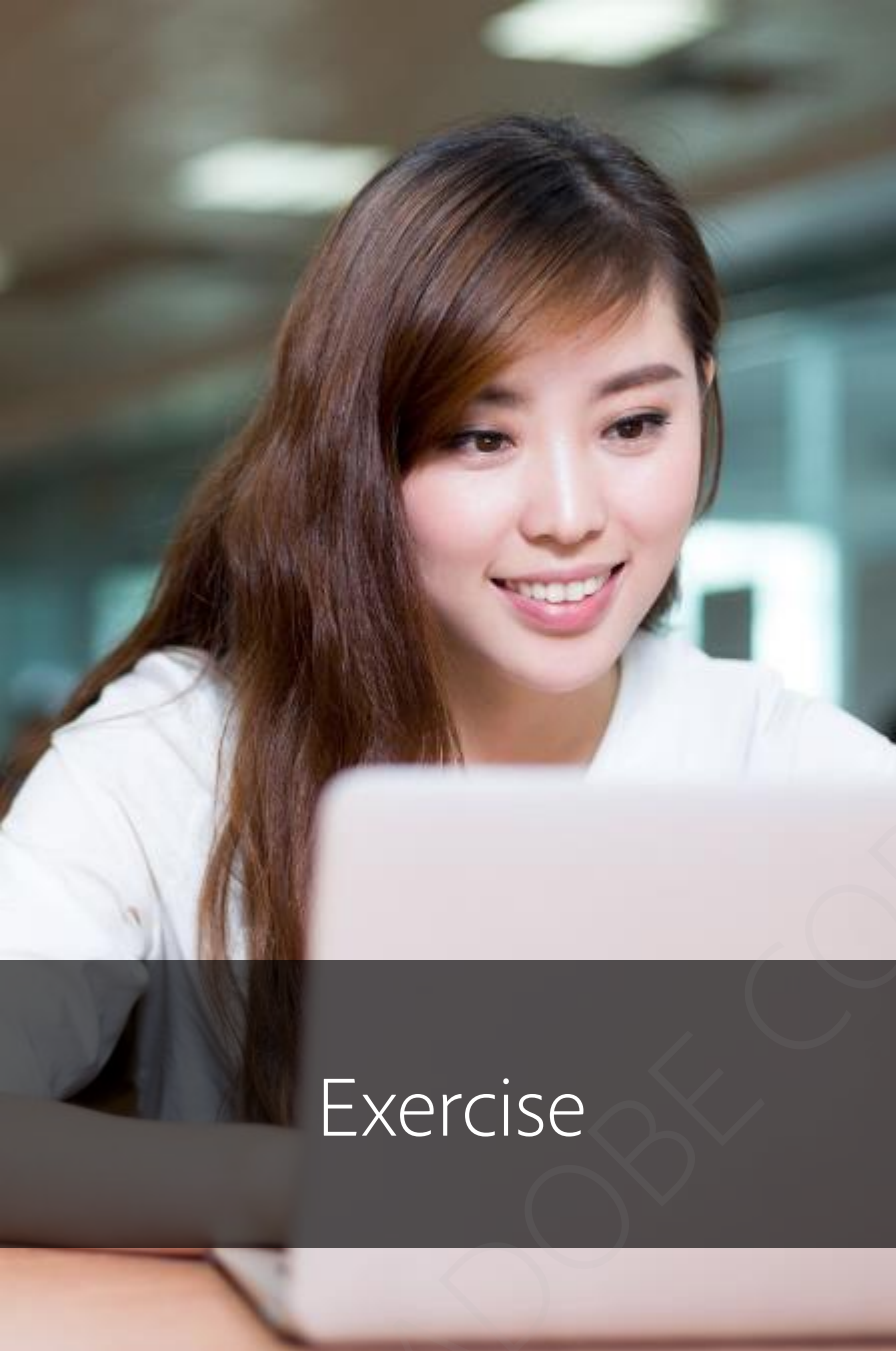
Import Stocks | Sling Listener

Give Business users the ability to create/remove stocks

- User creates a stock folder
- Changed resource is identified
- Symbol is extracted from the folder name
- A new scheduler config is created



Adding stocks to the page will be covered in the Sling Model module



Exercise

Exercise 3 - Create a resource listener

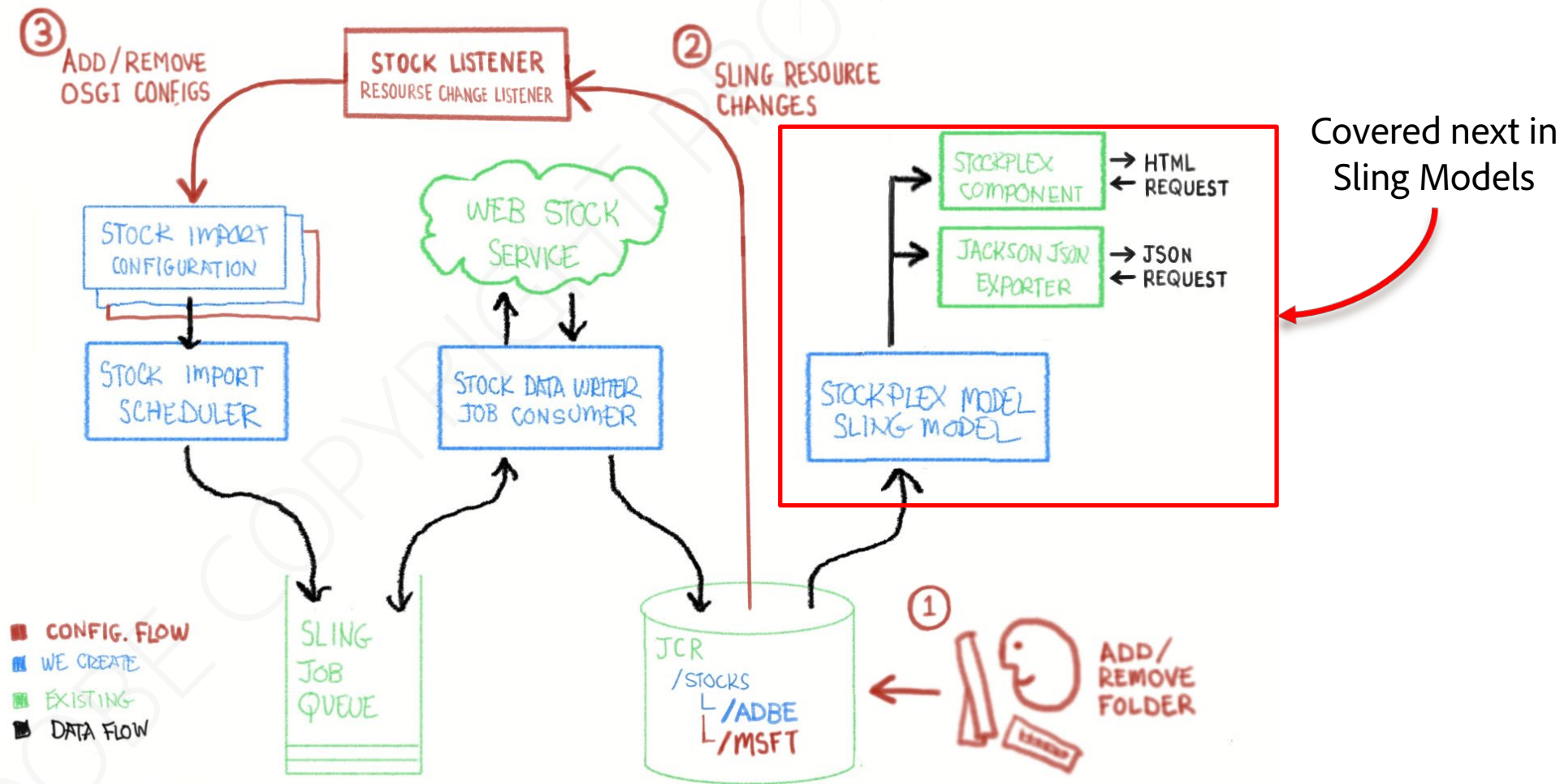
Scenario: An author goes to sites.html and creates a new Stock symbol folder under /content/stocks. This folder creates an OSGi config for the stock scheduler to import that stock symbol. If the author deletes a stock symbol folder, the OSGi config is removed.

ConfigurationAdmin service allows us to get/set OSGi configurations programmatically.

onChange() – listens for changes to /content/stocks

- If the created folder is not uppercase, this is autofixed
- If it was a folder added, a OSGi config is created
- If it was a folder deleted, a OSGi config is deleted

Use Case | Stock Import





Key Takeaways

Key takeaways from this module:

- OSGi Events:
 - Lightweight handlers that are used to:
 - Add a Sling Job to the JobManager
 - Trigger an AEM workflow
- Sling Scheduler
 - Enables you to easily schedule jobs within your application
- Event Types:
 - Node added, Node moved, Node removed, Property added, Property removed, and Property changed