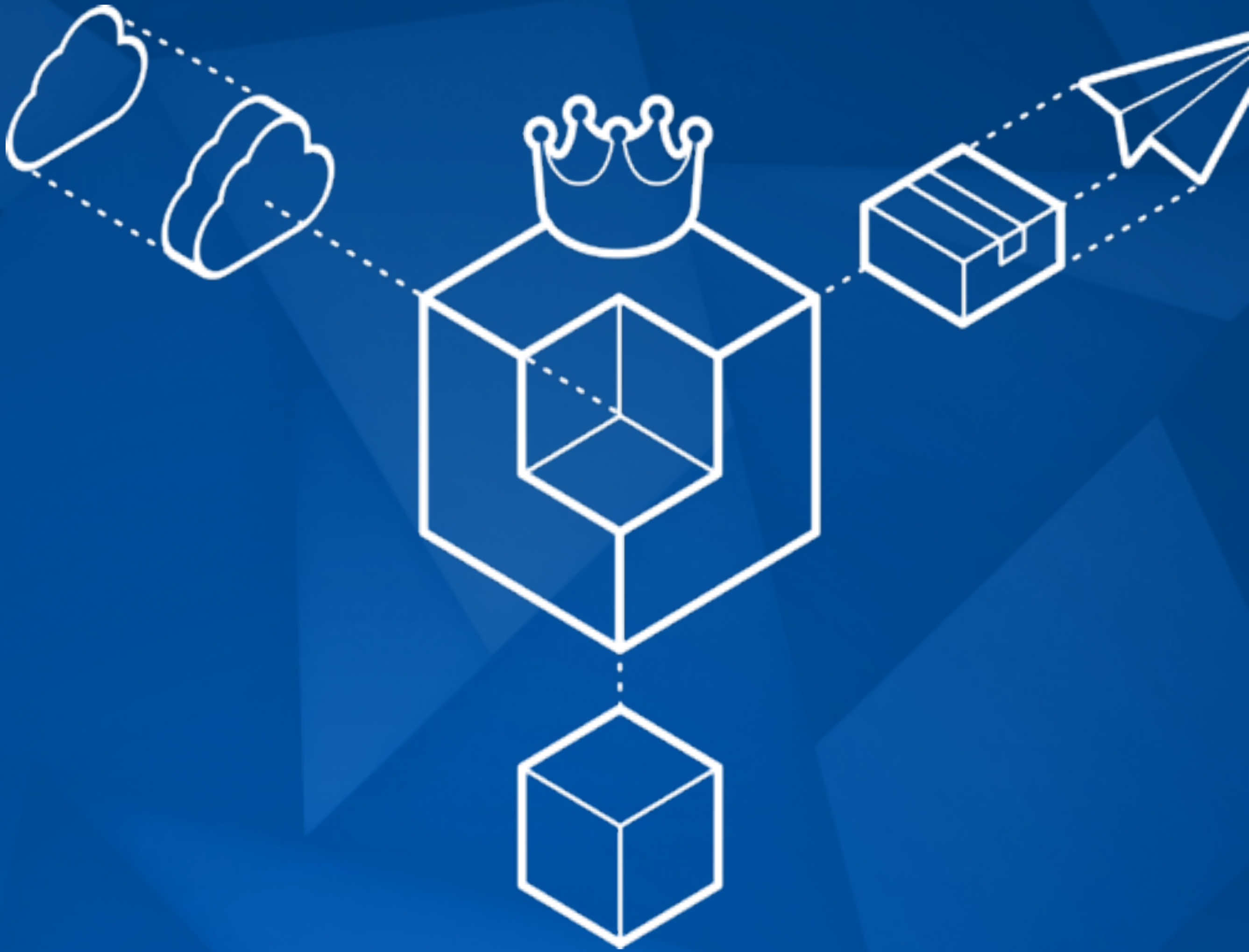
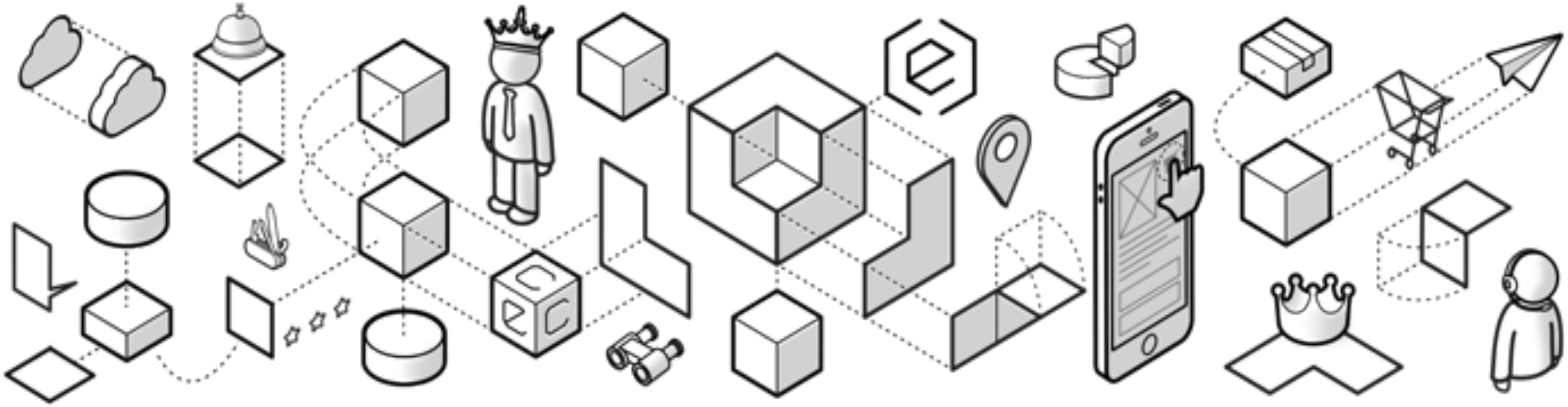
CronJobs





Configuring CronJobs

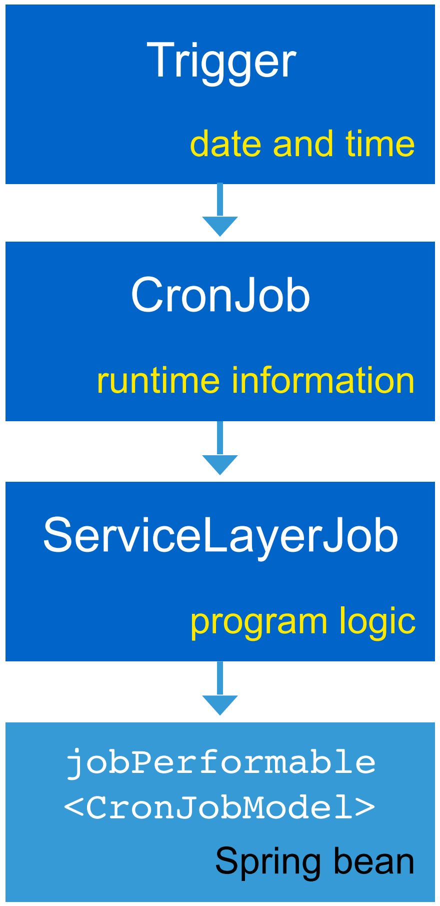
Configuring Cronjobs
Cronjob Scripting

CronJobs • Definition

- Automated tasks
- Performed at a certain time (such as 16:05), or at fixed intervals (such as every five minutes)
- Can be used for:
  - Backups
  - Updating / synchronizing catalog contents
  - Imports / Exports
  - Re-calculating prices
  - etc…

CronJobs • Key Facts

- A CronJob consists of a:
  - CronJob: Runtime information
  - Job: What to do
  - Trigger: When to run
- Allows re-using code and items
- CronJobs always run in a SessionContext (i.e. they have a user assigned)



Defining the Logic Behind the ServiceLayerJob

Step 1 • Define the logic

- Write a Java class implementing `JobPerformable<CronJobModel>` or extending `AbstractJobPerformable<CronJobModel>`

```
public class MyJob
   extends AbstractJobPerformable<CronJobModel>
   {
      public PerformResult perform(final CronJobModel cronJol
      {
         // Do something...
         return new PerformResult(CronJobResult.SUCCESS,
                                             CronJobStatus.FINISHED);
      }
   }
```

`Class Constants`

CronJobResult
ERROR
FAILURE
SUCCESS
UNKNOWN

CronJobStatus
ABORTED
FINISHED
PAUSED
RUNNING
RUNNINGSTART
UNKNOWN

Creating the ServiceLayerJob

Step 2 • Register a Spring Bean and create a ServiceLayerJob

- Configure `resource/extensionName-Spring.xml`

```
<bean id="myJob" class="my.bookstore.MyJob"
                parent="abstractJobPerformable"/>
```

Step 3 • Create a ServiceLayerJob item

- Running a system update (for essential data) will create a ServiceLayerJob item for each bean implementing `JobPerformable`
  - In our example, a ServiceLayerJob with code `myJob` will be created, referencing the `myJob` bean
- You can also create the ServiceLayerJob using ImpEx

```
INSERT_UPDATE ServicelayerJob;code[unique=true];springId
;myJob;myJob
```

Creating the CronJob and Trigger

Step 4 • Create a CronJob

- Use Backoffice, hMC, or ImpEx

```
INSERT_UPDATE CronJob;
code[unique=true];job(code);singleExecutable;sessionLanguage(isoc
ode)
;myCronJob;myJob;false;de
```

Step 5 • Create a Trigger

- Again, using Backoffice, hMC, or ImpEx
- The cronExpression below indicated to run this job every night at midnight

```
INSERT_UPDATE Trigger;cronjob(code)[unique=true];cronExpression
; myCronJob; 0 0 0 * * ?
```

- Basic format of a cron expression:
  - A * in any field means any value, ? means ignore. Year is optional.

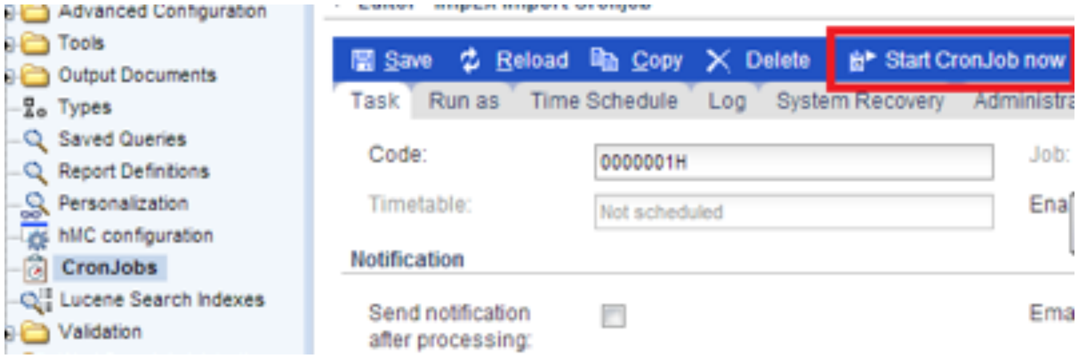
sec	min	hour	day	month	weekday	year
0-59	0-59	0-23	? 1-31	1-12	? 1-7	1970-2099

Overriding the Trigger • Starting the CronJob Explicitly

- Override schedule with ImpEx (for example, to start it immediately)

```
#%beanshell% afterEach: impex.getLastImportedItem().setActivationTime(new
Date());
```

- Or, using the hMC



Code: 0000001H
Timetable: Not scheduled
Notification
Send notification after processing: ☐

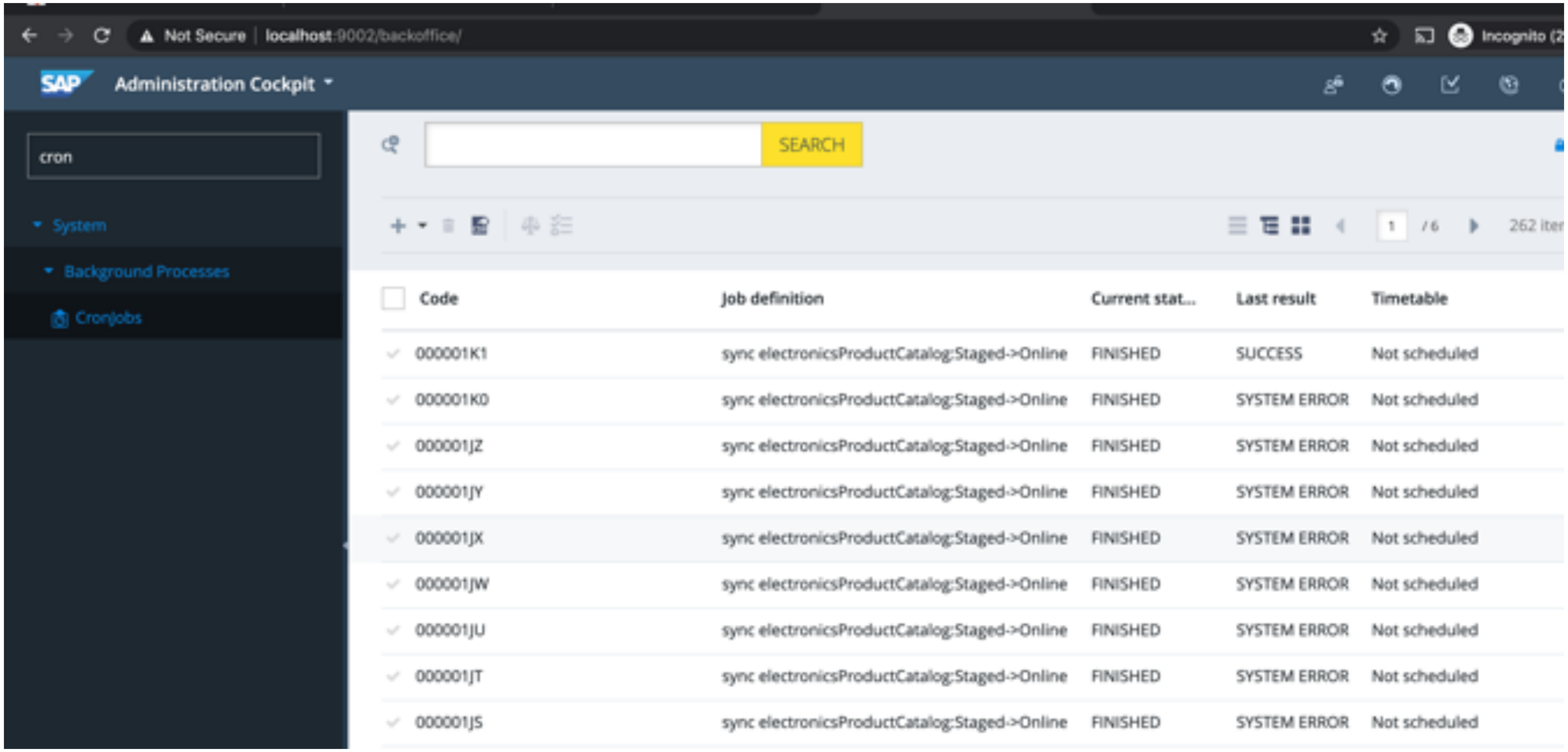
- Using Ant

```
ant runcronjob -Dcronjob=myCronJob
```

- Or directly from the API

```
cronJobService.performCronJob( myCronJobModel );
```

Cronjob in Backoffice



Code	Job definition	Current stat...	Last result	Timetable
000001K1	sync electronicsProductCatalog:Staged->Online	FINISHED	SUCCESS	Not scheduled
000001K0	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled
000001JZ	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled
000001JY	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled
000001JX	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled
000001JW	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled
000001JU	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled
000001JT	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled
000001JS	sync electronicsProductCatalog:Staged->Online	FINISHED	SYSTEM ERROR	Not scheduled

Useful CronJob properties

Email template

- notify certain user using given email template

Enable code execution

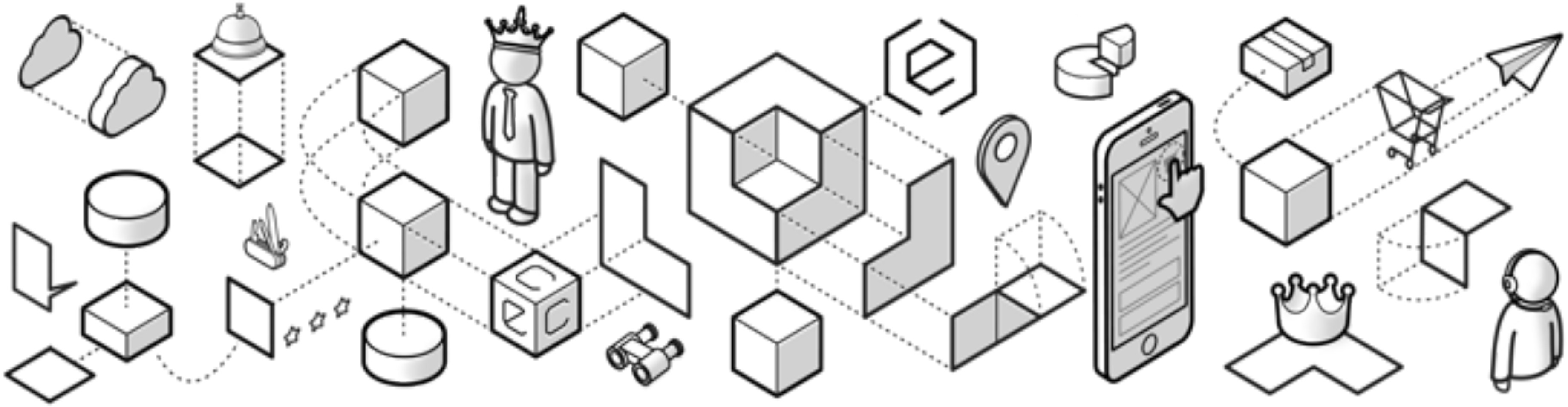
- enable or disable BeanShell, etc.

User

- to empower restrictions

Node id

- to specify server for job execution



CronJob Scripting

Configuring Cronjobs
Cronjob Scripting

Benefits of Cronjob Scripting

- Traditionally, creating a new CronJob is time-consuming and involves many manual steps:
  - Create a new java class for the Job
  - Define the new job as a Spring bean
  - Rebuild the code and restart the server

Using scripting, creating CronJobs becomes much easier and it can be done dynamically at runtime

CronJob Scripting API

- **Script** - the item type where the script content is stored

```
INSERT_UPDATE Script; code[unique=true];content
;myGroovyScript;println 'hello groovy! '+ new Date()
```

- **ScriptingJob** - subtype of ServicelayerJob, which contains the scriptURI (the script can be retrieved at runtime from classpath, DB etc.)

```
INSERT_UPDATE ScriptingJob; code[unique=true];scriptURI
;mydynamicJob;model://myGroovyScript
```

- **scriptingJobPerformable** - the implicit spring bean assigned to every **ScriptingJob** instance; it implements the usual **perform()** method.

Executing script-based CronJobs

- Creating a cronjob instance

```
INSERT_UPDATE CronJob; code[unique=true];job(code)
;mydynamicCronJob;mydynamicJob
```

- Executing a cronjob using a script

```
def dynamicCJ = cronJobService.getCronJob("mydynamicCronJob")
cronJobService.performCronJob(dynamicCJ,true)
```

- All other ways of execution can be used: Trigger, manual execution in hMC/Backoffice, and impex beanshell.
- In a script, you can return a cronjob result

```
println 'hello groovy! '+ new Date()
return new PerformResult(CronJobResult.SUCCESS,CronJobStatus.FI
NISHED)
```

CronJob Item as Context Parameter

- Context always contains the current **CronJobModel**
- It is passed as a context parameter (key="**cronjob**").

```
println 'hello groovy! '+ new Date()
println cronjob.code
println cronjob.status
```

Exercise 9