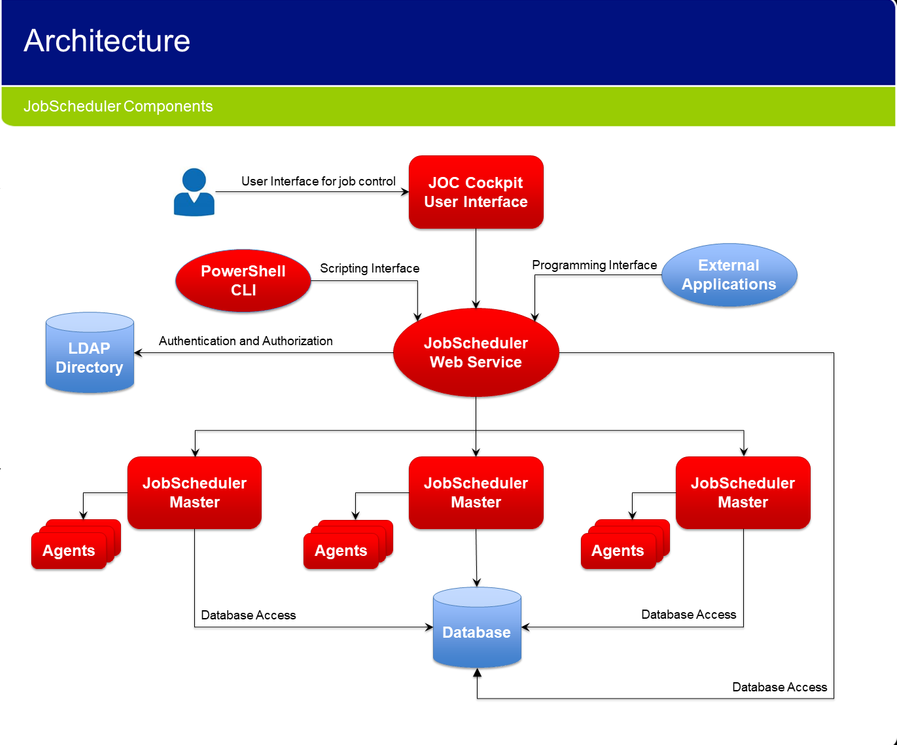
**SOS Berlin**

Index

1. Introduction
2. Architecture
3. Components In Details
4. Installation Guide
5. Features
6. Jobs,Job Chains,Standalone jobs,Process Class
7. JITL Jobs And APIs
8. Yade
9. Calendar
10. File Watcher and Directory Monitoring
11. Job Streams
12. Audit Log
13. Configuration
14. History
15. Resources And Blacklisted Resources
16. Email Notification
17. Roles And Permissions
18. **INTRODUCTION**

* SOS Berlin is a job scheduler tool which is mainly used for workload automation
* Job scheduler mainly consist of three main components JOC Cockpit, Master and Universal Agent
* It is an open source scheduling solution for enterprise level process automation
* It can be used to launch executable files and shell scripts and automatically run database procedures.
* It stores all the information in a back-end database.
* Job scheduler provides sequential and parallel tasks, job chains, distributed processing, events, job synchronization, clustering fail-over and a real API.
* It can be used as a load balancing with multiple masters and agent configurations.
* It is a highly scalable tool which also provide atomicity and durability.

1. **ARCHITECTURE**



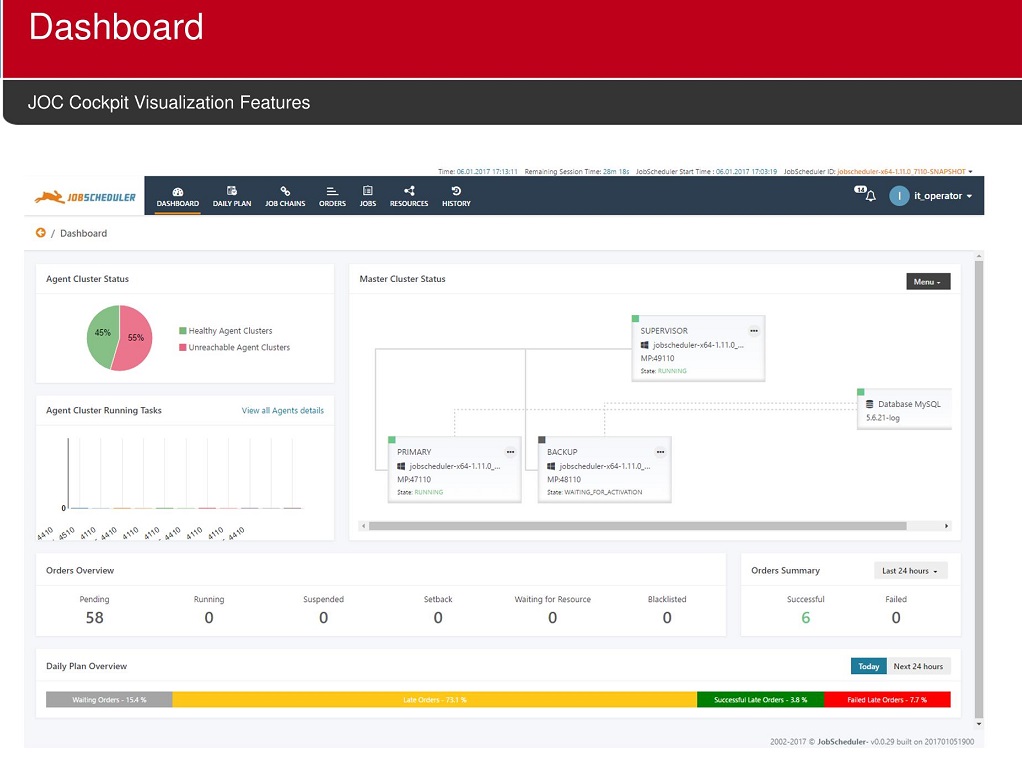
The architecture of SOS Berlin revolves around three main components

1. JOC Cockpit
2. Master
3. Agent

* JOC Cockpit is the user interface which acts as the front end for communicating with the master and the agent.
* JOC requires an authentication to log into the dashboard, it authenticates the user via job scheduler web service.
* The user information is stored in the `shiro.ini` file which is a default directory for user storage. It can also be configured with LDAP.
* The job scheduler web service also communicates with the database to fetch all the jobs and job chains.
* The master is the heart of the architecture which controls the agent and all the jobs
* Multiple master can be configured to create a cluster.
* The master also controls the agent and there can be multiple agents for a master.
* Master executes tasks and orchestrates agents. It can execute local tasks and well as remote tasks
* Agents are deployed on top of existing servers and can be accessed by any master.

1. **COMPONENTS IN DETAIL**

JOC COCKPIT



* Job Scheduler Operations Center(JOC) is the end user interface for job scheduler.
* JOC brings a modern, responsive user interface, web service API and finely configurable authentication and authorization to open source job scheduling.
* It provides a single interface for carrying out all the tasks from monitoring jobs and job chains to open source job scheduling.
* It is installed separately to a job scheduler master. This separation brings better user experience for service operators and allows use of the job scheduler in security sensitive and critical networks.
* JOC can be operated in less restrictive network zones than job scheduler master and agent.
* JOC give a lot of information such as the agent cluster status which shows which agents are available and which are not.
* It gives us the status of master whether it is running or not, it also shows the total number of orders and jobs and file transfers. In short it give us the whole summary.
* We can create a job,job chain,order via JOC. It also provides us with the calendar frequency.
* We can create our custom calendar and assign the job a specific calendar.
* We can view the audit logs and history of jobs executed.

MASTER

* Master is the heart of the SOS berlin architecture
* Master controls all the agent and is responsible to execute the tasks
* It executes all the local tasks as well as the remote tasks.
* It orchestrates the agents and works with them hand in hand.
* We can set up an active or passive master agent cluster.

AGENT

* Agent acts as an slave which obeys all the commands of the master.
* Agent can reside in the same machine where the master is installed or any other machine.
* Agent can be applied to a job via process class. It is a special type of class which is created for a particular agent and the job runs in the agent system.

1. **INSTALLATION GUIDE**

Note :

* Mater should be installed first and then you can proceed with the next components.
* Installation can be done in two ways with dialog boxes or without dialog boxes.
* To install the components without the dialog you will have to fill in all the details of the jobscheduler\_install.xml.
* After filling in the details you have to navigate the command prompt or the terminal where the setup.cmd or setup.sh resides and run the following command `setup.cmd `path\_of\_jobscheduler\_install.xml/jobscheduler\_install.xml`.
* To install it with dialog box just run the setup.cmd or setup.sh and follow the instructions.

MASTER

Windows Installation Guide/Linux Installation Guide

* Download the master windows/linux from <https://www.sos-berlin.com/en/jobscheduler-downloads>
* Choose the full install option and extract the downloaded file into an arbitrary location.
* Open the command prompt and navigate it to the extracted folder and open the setup.cmd/setup.sh, do not open the command prompt in administrator mode.
* Follow the instructions carefully, you can change all the default ports and folder structures as your requirements you can also setup your own SMTP server.
* You can choose between various databases such as postgre,mysql,oracle,h2,mariadb .If you find any problem in the databases connectivity please check if the database connectivity jar is present in the `lib` folder of the job scheduler.
* After successful connection the job scheduler will install a service which will run automatically. You can check your service in the local services section of your system.
* Make sure the service is running else the master will not work.
* After the installation the master creates all the necessary tables in the database which will be required by the JOC and Agent.

JOC

Windows Installation Guide/Linux Installation Guide

* Download the JOC windows/linux from <https://www.sos-berlin.com/en/jobscheduler-downloads>
* Choose the full install option and extract the downloaded file into an arbitrary location.
* Open the command prompt and navigate it to the extracted folder and open the setup.cmd/setup.sh, do not open the command prompt in administrator mode.
* Follow the instructions carefully and choose the same database which was selected while installing the master else the setup will fail.
* After successful connection the job scheduler will install a service which will run automatically. You can check your service in the local services section of your system.
* Make sure the service is running else the JOC will not work.
* After the installation the JOC will be available on port 4446 if you did not change the default port you can navigate to <http://localhost:4446> to view JOC.
* The default username and password is root.

AGENT

Windows Installation Guide/Linux Installation Guide

* Download the Agent windows/linux from <https://www.sos-berlin.com/en/jobscheduler-downloads>
* Choose the full install option and extract the downloaded file into an arbitrary location.
* Open the command prompt and navigate it to the extracted folder and open the setup.cmd/setup.sh, do not open the command prompt in administrator mode.
* Follow the instructions carefully, agent does not require any special configurations.
* After successful installation a service will be created.
* Agent will be available on port 4445 if you did not change the default port.

1. **FEATURES**

SOS Berlin offers some additional features which proves handy in quite a lot of situations.

* Download JOC Log : you can download the whole JOC logs which is available in the dashboard panel of JOC. It contains the entire log of the jobs that was running in the scheduler.
* Edit Layout : You can edit the layout of the dashboard according to your requirements the edit layout option is present in the dashboard panel of the JOC.
* Export to Excel : You can export the summary of Job Chains,Jobs,Orders,Job Streams,History and Audit Logs into excel format. This provides the brief summary of the specific panel.
* Customization : You can customize the History and Audit Log panel according to your requirement.

1. **JOBS,JOB CHAINS,ORDER,PROCESS CLASS**

* Jobs can be of two types standalone job or ordered job.
* Jobs are stored with the `.job` file extension.
* Job is an individual piece of code which is used to perform some defined activities, it can be triggering a stored procedure,performing some database activities, running a software or doing some other activities.
* Job can be written in various languages such as shell script,java script,SQL or java.

Example of a job (Ordered)

<?xml version="1.0" encoding="ISO-8859-1" ?>

<job title="read file via shell" order="yes" stop\_on\_error="false">

<params>

<param name="param1" value="Value1-Job1"/>

<param name="param2" value="Value2-Job1"/>

</params>

<script language="shell"><![CDATA[

cd E:\File\_Folders\_Output

E:

hello\_world.bat

]]></script>

<run\_time/>

</job>

The above job is an ordered shell script job. Note that the `order=yes` in the <job title> tag. It consists of two parameters which are enclosed in the <params> tag and its triggers hello\_world.bat which resides in E:\File\_Folders\_Output.

Example of a job (Standalone)

<?xml version="1.0" encoding="ISO-8859-1" ?>

<job title="read file via shell" stop\_on\_error="false">

<params>

<param name="line\_execution\_number" value="1"/>

<param name="param2" value="Value2-Job1"/>

</params>

<script language="shell"><![CDATA[

cd E:\File\_Folders\_Output

E:

resume\_urt\_jobexecution.bat

]]></script>

<run\_time/>

</job>

The above job is the example of a standalone job. Note there is no `order=yes` property in the <job title> tag. This is a shell job which is used to trigger resume\_utr\_jobexecution.bat which resides in E:\File\_Folders\_Output.

* Order is a trigger which is used to initialize the job.
* Order is saved as `.order` extension
* Order can be with parameters or without parameters.
* A standalone job does not require order to starts its execution whereas ordered job requires order for execution.

Example of an order

<?xml version="1.0" encoding="ISO-8859-1" ?>

<order title="run utr\_utility\_fileWatacher">

<params>

<param name="command" value="\${param3}"/>

<param name="Script2Execute" value="dir E:"/>

</params>

<run\_time/>

</order>

The above example is of order with parameters.

* Job chain consist of multiple jobs in a chain which is executed one after another.
* Job chains are stored with `.job\_chain` extension.

Example of job chain

<?xml version="1.0" encoding="ISO-8859-1" ?>

<job\_chain>

<file\_order\_source directory="E:\File\_Folders\_Output" regex="^utr\_no\*\.txt$|^utr\_no\_2\*\.txt$"/>

<job\_chain\_node job="readFileShell2" next\_state="insertDb" error\_state="error" state="readFile"/>

<job\_chain\_node job="Mysql\_JOB" next\_state="postProcessing" error\_state="error" state="insertDb"/>

<job\_chain\_node job="post\_processing\_job" next\_state="success" error\_state="error" state="postProcessing"/>

<job\_chain\_node state="success"/>

<job\_chain\_node job="post\_processing\_job\_error" error\_state="finalError" state="error"/>

<job\_chain\_node state="finalError"/>

</job\_chain>

The above example is of a job chain. Note the `next\_state` property which points to the `state` property of the job and also note the `error` property which points to the state `error`

* Process Class is the agent it is created automatically after you add an agent into the cluster however you can edit the process class according to your requirement.
* Process Class when applied to a job will trigger the code in the agent system.
* Process class is stored with `.process\_class` extension.
* Process class does not contain any extra code all the configuration resides in the JOC Cockpit.

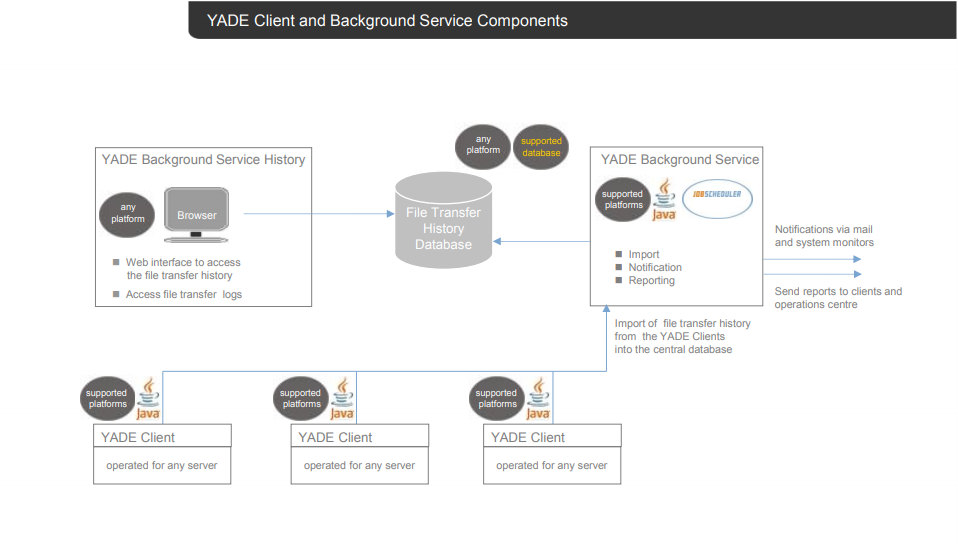
1. **JITL JOBS AND APIs**

* JITL ( JobScheduler Integrated Template Library ) is a set of standard jobs that is shipped with the job scheduler and can be easily parameterized for you environment.
* These are the predefined job which can be further manipulated according to your requirements
* You can further explore the JITL and find some examples in the link below

[https://kb.sos-berlin.com/display/PKB/JITL+Jobs+by+topic#JITLJobsbytopic-file\_operationsFileOperations](https://kb.sos-berlin.com/display/PKB/JITL+Jobs+by+topic" \l "JITLJobsbytopic-file_operationsFileOperations)

Following are the list of JITL jobs templates

1. File Operations : This template consist of file operations such as read,write, copy, exists and remove.
2. Managed File Transfer : This includes all the file transfer operations, it involves YADE.
3. Agentless Scheduling : job for execution of programs and scripts on remote SSH servers.
4. Installation : this involves the agent batch installer and JSBatchInstaller.
5. Operation : this involves operations such as checking the blacklisted resource,check run history,create daily schedule.
6. Housekeeping : it involves cleanup of files,rotate log, dequeue mail job, job scheduler restart.
7. Job Chain Control : this involves job chain splitter, synchronize job chains, report all parameters.
8. E-Mail : this involves sending mail, read inbox , process inbox.
9. Eventing : this includes the event job , dequeue event job, check exist event, submit event job.
10. Database : This includes the various database job , this includes all the databases which is compatible with SOS berlin.
11. Migration / Conversion : this includes the job scheduler cron adapter.
12. Other : this includes the job not covered in the above sections which are remote control command job,text processor,xls transform,http post job etc.
13. **YADE**



* YADE is used for file transfer operation, this file transfer can happen between master and agent or within the master
* You can download the Yade CLI from the link below

<https://www.sos-berlin.com/en/yade-downloads>

* YADE has many components such as YADE client which includes the command line interface, YADE APIs and JITL jobs.
* YADE background service includes the background service jobs, background service history and background service reporting.

FEATURES

* Send and receive files by FTP to/from some target host.
* Send and receive files by SFTP to/from some target host.
* Execute commands by SSH.
* Password and public/private key authentication are supported for SFTP.
* The parameterization is effected by command line parameters and/or by configuration file.
* Polling can also be done using YADE.

YADE has a configuration file which is used to provide the source and destination paths and also the file name which is to be transferred.

You can use regex patterns to provide the file name and in case you want to transfer multiple files.

<?xml version="1.0" encoding="ISO-8859-1" ?>

<job title="transferring the file from one folder to ano">

<script language="shell"><![CDATA[

cd C:\Program Files\sos-berlin.com\jade\client\bin

jade.cmd -settings=E:\File\_Folders\_Output\Yade\jade\_settings\_windows.ini -profile="localhost\_copy"

]]></script>

<run\_time/>

</job>

* The above example shows a YADE job.
* It redirects to the path where the jade.cmd is present and following that the `-settings=<path of jade\_settings\_windows.ini>` and `-profile=”localhost\_copy”`
* The `settings` property states the .ini file where the configurations are done.
* The `profile` property states which operation should be performed

Note: The profile can be found within `[ ]` in the jade\_settings\_windows.ini file

[globals]

write\_enable=YES

[BackgroundService]

BackgroundServiceHost = localhost

BackgroundServicePort = 4444

SendTransferHistory = true

TransferMethod = TCP

[MailSettings]

file\_notification\_to = myemail@mydomain.com

file\_notification\_subject = YADE Client mail

file\_notification\_body = This is a test.

mail\_smtp = mail.myemail.com

[jadeHistory]

log\_filename = ${HistFileLocation}/jade\_history.log

HistoryFileName = ${HistFileLocation}/jade\_history.csv

[ftp\_server\_2\_local]

;;

operation = copy

;;

source\_host = test.sos-berlin.com

source\_protocol = ftp

source\_port = 21

source\_user = demo

source\_password = demo

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[localhost\_copy]

include = globals, jadeHistory , BackgroundService

source\_host = localhost

source\_protocol= local

target\_host = localhost

target\_protocol= local

file\_spec = ^.\*\.txt$

operation = copy

source\_dir = E:\File\_Folders\_Output\Yade\a

target\_dir = E:\File\_Folders\_Output\Yade\b

poll\_interval = 60

poll\_minfiles = 1

poll\_timeout = 1

[localhost\_send]

include = globals

;;

operation = copy

file\_spec = ^.\*\.txt$

;;

source\_host = localhost

source\_protocol= local

source\_dir = E:\File\_Folders\_Output\Yade\a

;;

target\_host = test.sos-berlin.com

target\_port = 21

target\_protocol= FTP

target\_user = demo

target\_password= demo

target\_dir = /

[ftp\_server\_2\_local\_pass]

;;

operation = copy

;;

source\_host = test.sos-berlin.com

source\_protocol = ftp

source\_port = 21

source\_user = demo

source\_password = demo

source\_dir = E:\File\_Folders\_Output\Yade

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[sftp\_server\_2\_local\_pass]

;;

operation = copy

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_ssh\_auth\_method = password

source\_port = 22

source\_user = demo

source\_password = demo

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[ftp\_server\_2\_local\_select\_recursive]

;;

operation = copy

file\_spec = ^test\_.\.txt$

;;

recursive = true

;;force\_files = false

;;

source\_host = test.sos-berlin.com

source\_protocol = ftp

source\_port = 21

source\_user = demo

source\_password = demo

source\_dir = /rec

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[sftp\_server\_2\_local\_select\_recursive]

;;

operation = copy

file\_spec = ^test\_.\.txt$

;;

recursive = true

;;force\_files = false

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_ssh\_auth\_method = password

source\_port = 22

source\_user = demo

source\_password = demo

source\_dir = /rec

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[ftp\_server\_2\_local\_zero\_byte]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;file\_spec = ^test\_[4]\.txt$

;;

;;zero\_byte\_transfer = no

zero\_byte\_transfer = relaxed

;;zero\_byte\_transfer = strict

;;

source\_host = test.sos-berlin.com

source\_protocol = ftp

source\_port = 21

source\_user = demo

source\_password = demo

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[sftp\_server\_2\_local\_zero\_byte]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;file\_spec = ^test\_[4]\.txt$

;;

;;zero\_byte\_transfer = no

zero\_byte\_transfer = relaxed

;;zero\_byte\_transfer = strict

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_ssh\_auth\_method = password

source\_port = 22

source\_user = demo

source\_password = demo

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[ftp\_server\_2\_local\_poll\_minfiles]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;

poll\_interval = 20

poll\_timeout = 1

poll\_minfiles = 3

poll\_keep\_connection = true

;;

source\_host = test.sos-berlin.com

source\_protocol = ftp

source\_port = 21

source\_user = demo

source\_password = demo

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[sftp\_server\_2\_local\_poll\_minfiles]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;

poll\_interval = 20

poll\_timeout = 1

poll\_minfiles = 3

poll\_keep\_connection = true

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_ssh\_auth\_method = password

source\_port = 22

source\_user = demo

source\_password = demo

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[replace\_local\_datestamp]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;

replacing = ^(test)(\_)[0-9]\.txt

replacement = file\_;[date:yyyy-MM-dd]\_

;;

source\_host = localhost

source\_protocol = local

source\_dir = E:\File\_Folders\_Output\Yade\a

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\b

[local\_2\_local\_replace\_datestamp]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;

replacing = ^(test)(\_)[0-9]\.txt

replacement = file\_;[date:yyyy-MM-dd]\_

;;

source\_host = localhost

source\_protocol = local

source\_dir = E:\File\_Folders\_Output\Yade\a

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\b

[sftp\_server\_2\_local\_cumulate]

;;

operation = copy

file\_spec = ^test\_.\.txt$

;;

cumulate\_files = true

cumulative\_file\_name = text-files.txt

cumulative\_file\_separator = --- File: %{SourceFileName} ---

;;cumulative\_file\_delete = true

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_ssh\_auth\_method = password

source\_port = 22

source\_user = demo

source\_password = demo

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[local\_2\_local\_create\_md5]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;

CreateSecurityHashFile = true

;;

source\_host = localhost

source\_protocol = local

source\_dir = E:\File\_Folders\_Output\Yade\a

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\b

[local\_2\_local\_check\_md5]

;;

operation = copy

file\_spec = ^test\_[0-9]\.txt$

;;

CheckSecurityHash = true

;;

source\_host = localhost

source\_protocol = local

source\_dir = E:\File\_Folders\_Output\Yade\b

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\b\checked

[ftp\_server\_2\_local\_atomic]

;;

operation = copy

file\_spec = ^test\_large\_1\.txt$

;;

atomic\_prefix = ~

atomic\_suffix = ~

;;

source\_host = test.sos-berlin.com

source\_protocol = ftp

source\_port = 21

source\_user = demo

source\_password = demo

source\_dir = /large/

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[sftp\_server\_2\_local\_atomic]

;;

operation = copy

file\_spec = ^test\_large\_1\.txt$

;;

atomic\_prefix = ~

atomic\_suffix = ~

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_ssh\_auth\_method = password

source\_port = 22

source\_user = demo

source\_password = demo

source\_dir = /large/

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[local\_2\_local\_steady\_state]

;;

operation = copy

file\_spec = ^test\_large\_1\.txt$

;;

check\_steady\_state\_of\_files = true

steady\_state\_count = 10

check\_steady\_state\_interval = 2

;;

source\_host = localhost

source\_protocol = local

source\_dir = E:\File\_Folders\_Output\Yade\a

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\b

[local\_2\_local\_check\_steady\_state]

;;

operation = copy

file\_spec = ^test\_large\_1\.txt$

;;

check\_steady\_state\_of\_files = true

steady\_state\_count = 10

check\_steady\_state\_interval = 2

;;

source\_host = localhost

source\_protocol = local

source\_dir = E:\File\_Folders\_Output\Yade\a

;;

target\_host = localhost

target\_protocol = local

target\_dir = ${USERPROFILE}\jade\_demo\b

[sftp\_server\_2\_local\_key]

;;

operation = copy

file\_spec = ^test\.txt$

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_port = 22

source\_user = demo

source\_ssh\_auth\_method = publickey

source\_ssh\_auth\_file = E:\File\_Folders\_Output\Yade\demo\_id\_rsa

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

target\_dir = E:\File\_Folders\_Output\Yade\a

[sftp\_server\_2\_local\_pub\_priv\_key]

;;

operation = copy

file\_spec = .\*

;;

source\_host = test.sos-berlin.com

source\_protocol = sftp

source\_port = 22

source\_user = demo

source\_ssh\_auth\_method = publickey

source\_ssh\_auth\_file = E:\File\_Folders\_Output\Yade\demo\_id\_rsa

source\_dir = /

;;

target\_host = localhost

target\_protocol = local

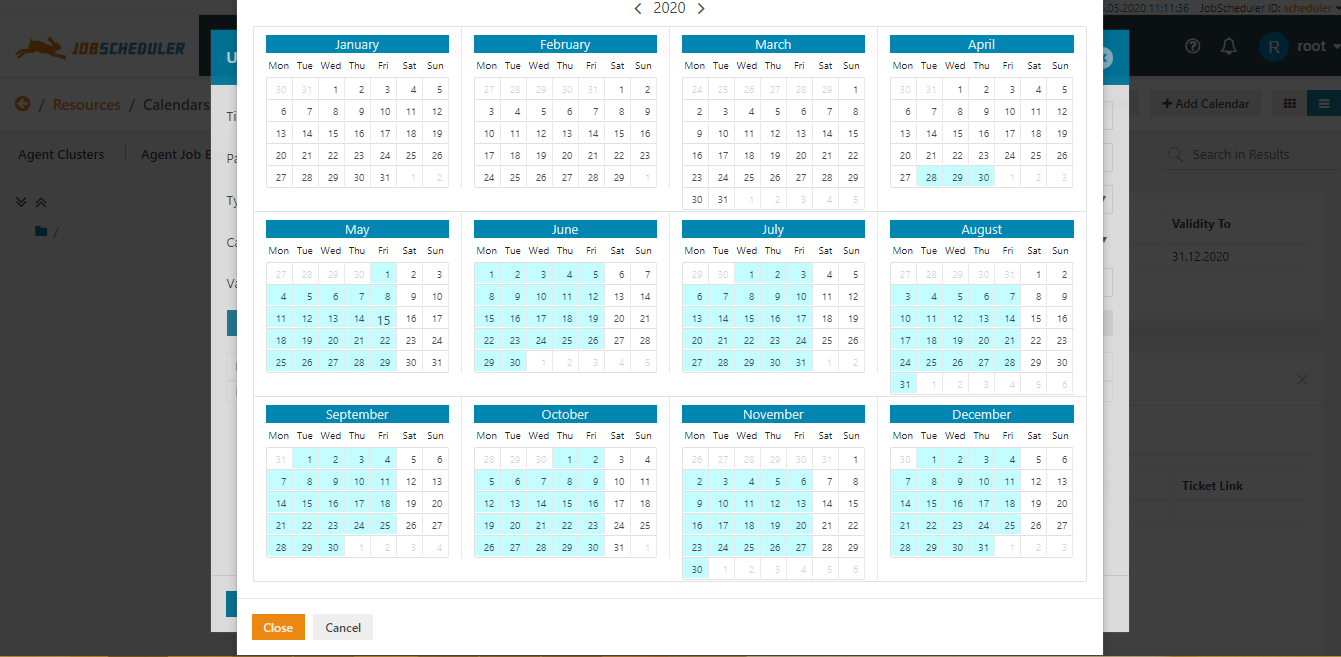
target\_dir = E:\File\_Folders\_Output\Yade\a

The above code is the example of jade\_settings\_windows.ini file.

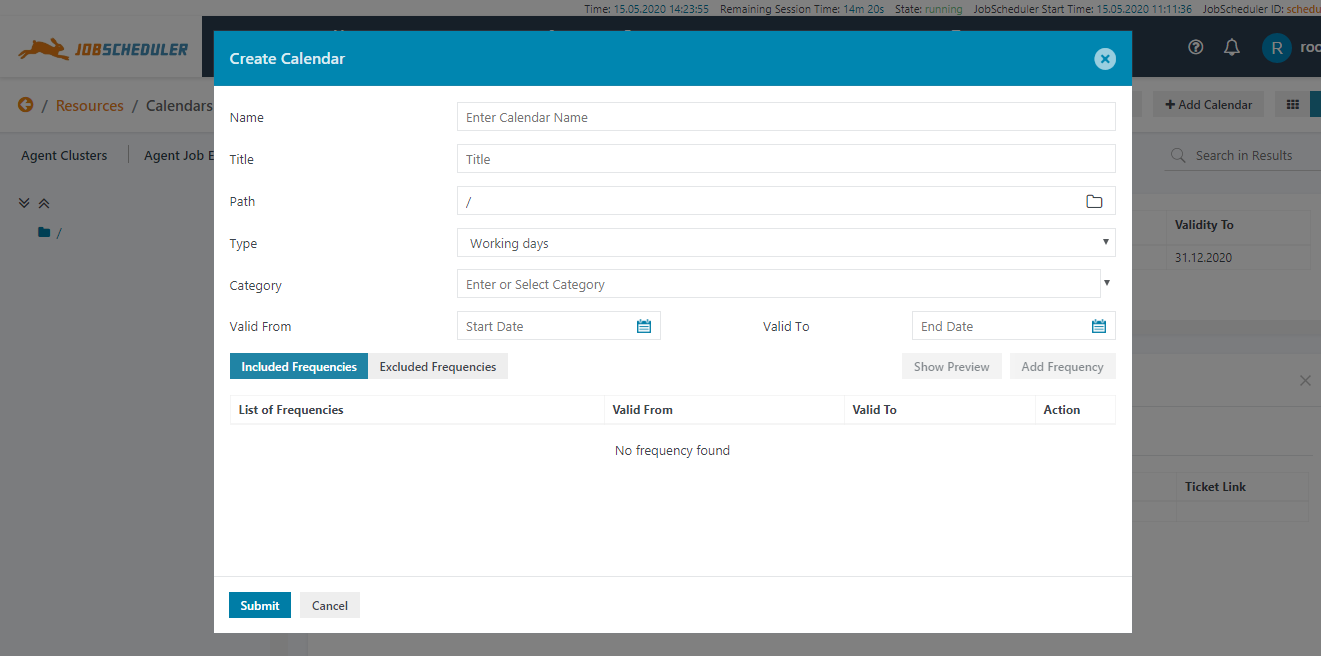
1. **CALENDAR**

SOS Berlin provides us with the calendar feature which can be used to schedule the jobs according to the calendar dates.

We can customize our calendar and apply it to the job.



* Calendar is available on the JOC Cockpit under the `Resources` tab.
* We can add a new calendar by clicking on the `Add Calendar` button.
* We can also import a calendar using a import calendar feature. The file accepted is JSON.
* After clicking on the add calendar feature a pop appears



* You have to fill in all the required details.
* The included frequency means all the days which you want your job to run.
* The excluded frequency means all the days which you want your job to not run.
* You can apply this calendar to a job by clicking on the action button on the left side of the job and then click on the set runtime option and click on the assign calendar button, your created calendar will appear on the screen.

1. **FILE WATCHER AND DIRECTORY MONITORING**

* File watcher is used to automatically start jobs and job chains when files arrive in a specified folder.
* File watching is performed on the Job Scheduler master or agent
* A file order is created for each file that arrives in the folder being watched and this order is used to start a job in job chain.
* Multiple folders can be watched and filters can be applied(regular expressions).
* The last stage of a file order requires that the file that originated in the folder is moved from the directory being watched.It provides the file sinks that will automatically move or remove the file.

<?xml version="1.0" encoding="ISO-8859-1" ?>

<job\_chain>

<file\_order\_source directory="E:\File\_Folders\_Output" regex="^utr\_no\*\.txt$|^utr\_no\_2\*\.txt$"/>

<job\_chain\_node job="readFileShell2" next\_state="insertDb" error\_state="error" state="readFile"/>

<job\_chain\_node job="Mysql\_JOB" next\_state="postProcessing" error\_state="error" state="insertDb"/>

<job\_chain\_node job="post\_processing\_job" next\_state="success" error\_state="error" state="postProcessing"/>

<job\_chain\_node state="success"/>

<job\_chain\_node job="post\_processing\_job\_error" error\_state="finalError" state="error"/>

<job\_chain\_node state="finalError"/>

</job\_chain>

In the above example note the `<file\_order\_source>` tag this is a file watcher the `directory` property includes the folder that should be watched the `regex` property is a filter(regular expression) which only watches the files that match it.

* Directory Monitoring is similar to file watcher but the code is little bit different.
* Job scheduler can monitor file creation with a (standalone) job and with an order and a job chain.

<job name = "scheduler\_file\_notification\_1">

<script language="shell">

<![CDATA[

                set

                echo %SCHEDULER\_JOB\_NAME%

                echo %SCHEDULER\_TASK\_TRIGGER\_FILES%

                rem del %SCHEDULER\_TASK\_TRIGGER\_FILES%

            ]]>

</script>

<start\_when\_directory\_changed directory = "c:/temp" regex = "^file[1|2]\.xml$"/>

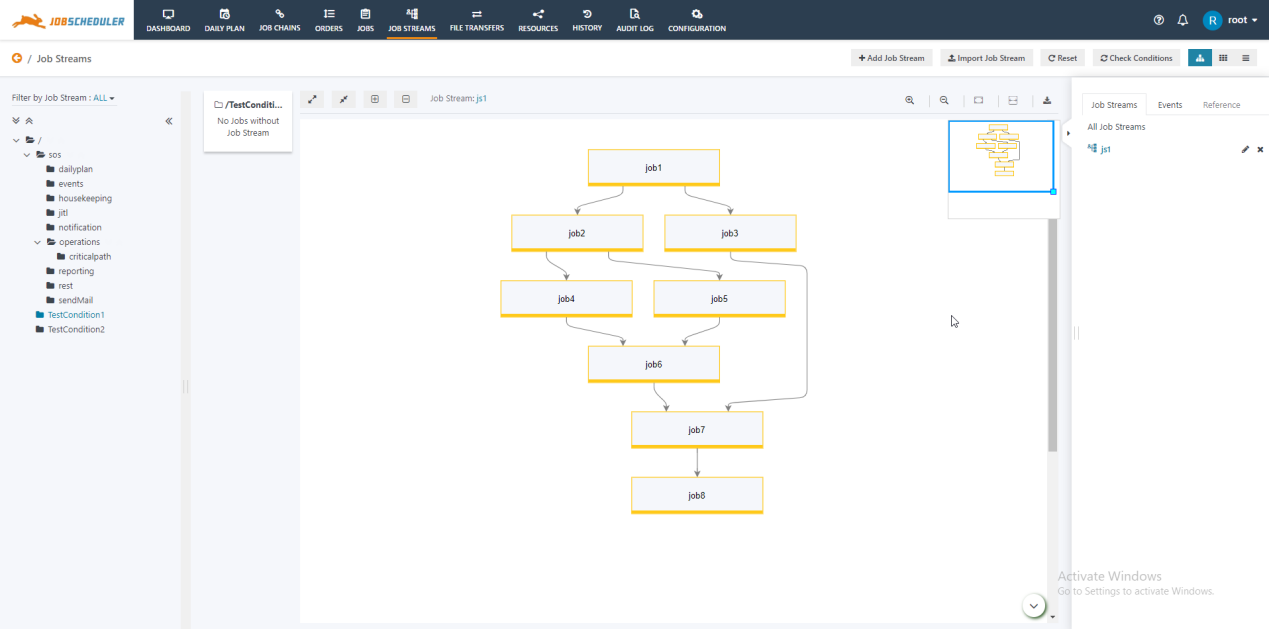
</job>

The above example shows the directory monitoring note the `<start\_when\_directory\_changed>` this is the difference between file watcher and directory monitoring.

Rest of the property is same as file watcher.

1. **JOB STREAMS**

* Job streams are mean to implement dependencies with job. This feature improves the existing forward dependencies(job chains) by introduction backward dependencies (conditions).
* Job streams bring a number of new objects to the user interfaces such as event ,in condition, out condition.
* A graphical representation of job streams is available on JOC Cockpit under the `Job Stream` tab.
* You can import a job stream or create it.

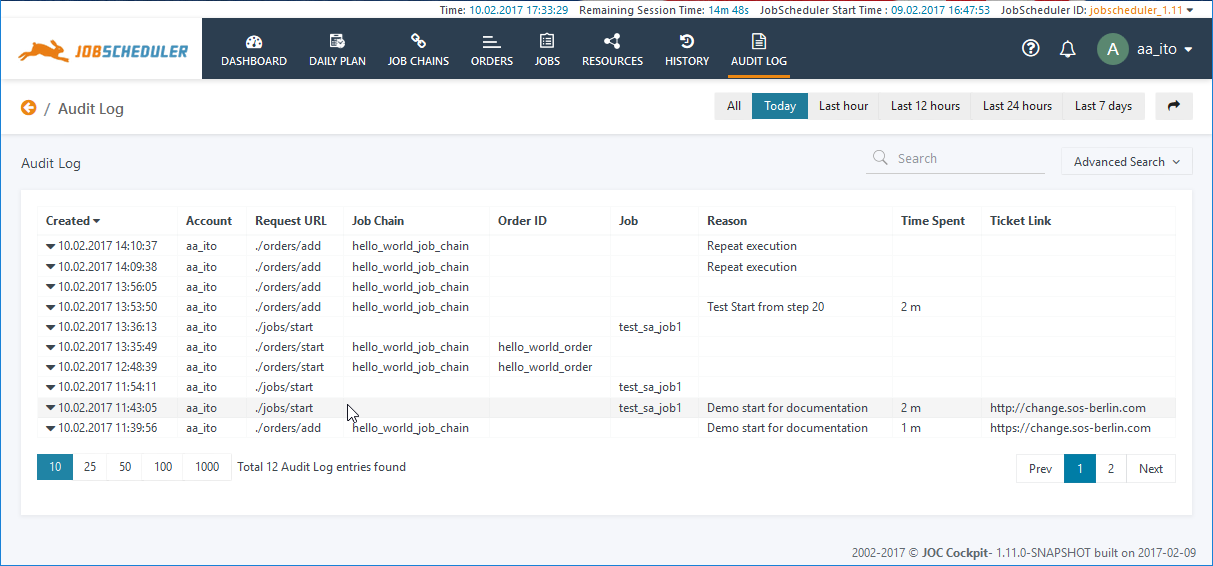


As you can see in the representation above shows the flow of the job from one to another.

Exit and error condition can also be added as well as success condition.

1. **AUDIT LOG**

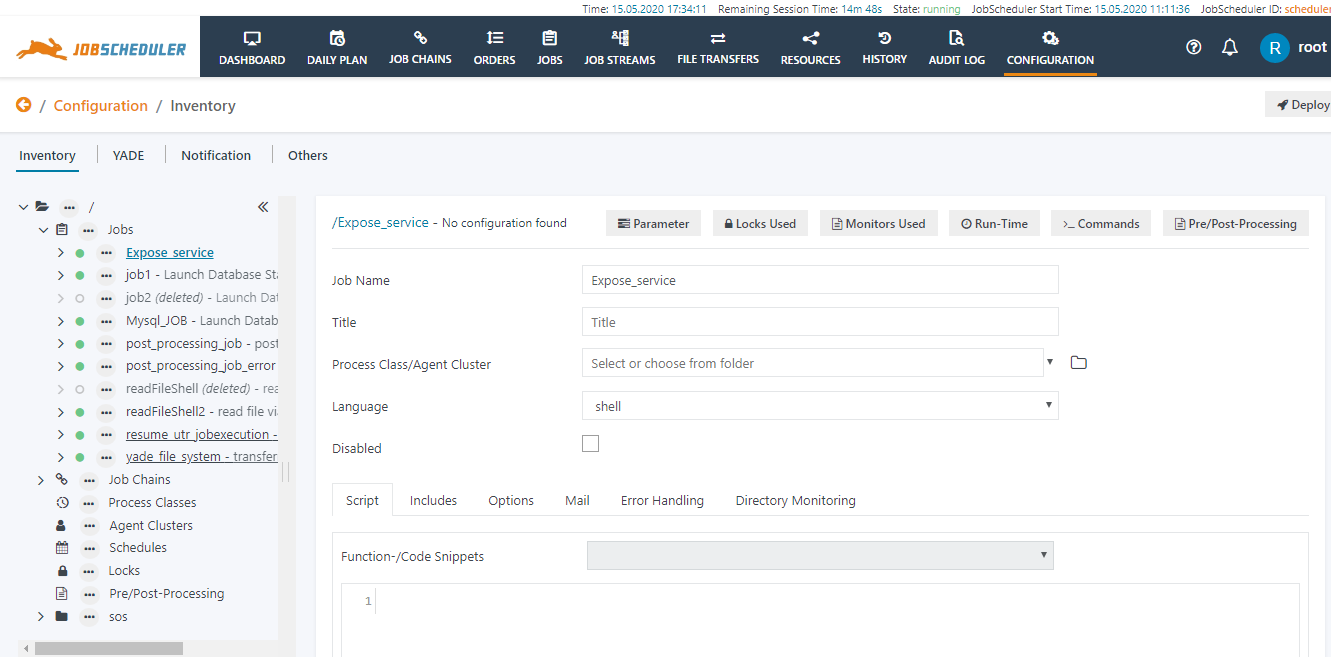
* Audit log provides us with the logs of the jobs that had been executed.
* Audit Log tab can be found in the JOC Cockpit.
* The logs can be downloaded and the summary can be exported to excel.



You can select the logs date wise and also customize base on your requirements.

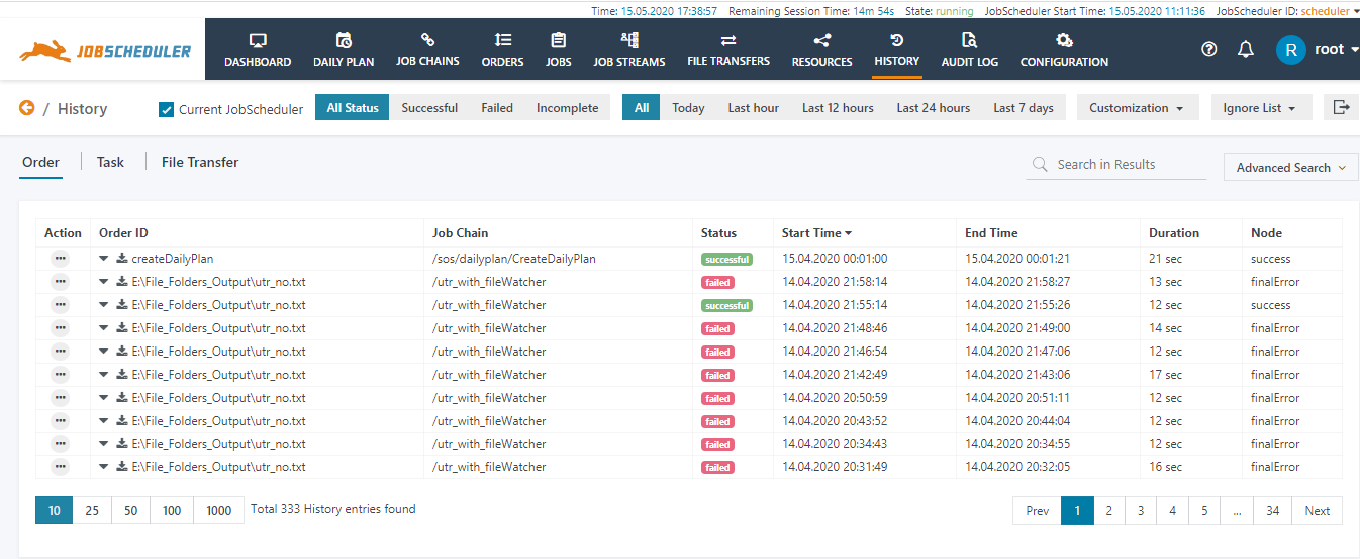
1. **CONFIGURATION**

* The Configuration panel is available in JOC Cockpit
* Configuration allows us to create and deploy job, job chains, orders and process class.
* It allows us to create agent cluster.
* You can also edit the jobs created.



1. **HISTORY**

* History panel is available in JOC Cockpit
* History panel shows all the jobs that have been run date and time wise
* You can export the summary to excel sheet.
* You can customize to view history according to a specific dates.



1. **RESOURCES AND BLACKLISTED RESOURCES**

* Resources tab shows all the resources that is available on the master
* This provides a single instance to manage all the resources.
* Resources panel can also be customizes according to your requirements.
* You can export the summary of the resources to excel.
* Resources panel consist of the many sub tabs such as agent clusters, agent job executions,events, locks, process classes,schedules,calendars, documentation.

1. **EMAIL NOTIFICATION**

* You can send email notification via job scheduler.
* It has a built in JITL job which triggers mail to the user.
* Mail can be sent on success, failure or warning of a job.
* Attachment can also be sent with mail.

 <job  title="Dies ist der Titel des Jobs" stop\_on\_error="no" order="yes" name="email\_monitor">

           <settings >

               <log\_level ><![CDATA[debug9]]></log\_level>

           </settings>

           <params >

               <param  name="host" value="smtp.sos"/>

               <param  name="port" value="25"/>

               <param  name="from" value="scheduler@ur"/>

               <param  name="MailOnJobStart\_to" value="uwe.risse@sos-berlin.com"/>

               <param  name="MailOnJobStart\_subject" value="Job: %{SCHEDULER\_JOB\_NAME}  Task: %{SCHEDULER\_TASK\_ID}"/>

               <param  name="MailOnJobStart\_body" value="<html>   <head>      <meta http-equiv="content-type" content="text/html; charset=ISO-8859-15">     <title></title>   </head>   <body text="#000000" bgcolor="#FFFFFF">     <b>Scheduler ID:</b> &nbsp; &nbsp;&nbsp; &nbsp;&nbsp;&nbsp;     %{SCHEDULER\_ID}<br>     <b>Scheduler</b>:&nbsp;&nbsp;&nbsp; &nbsp;&nbsp; &nbsp;     &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;     %{SCHEDULER\_HOST}:%{SCHEDULER\_TCP\_PORT}<br>     <b>Configuration in:</b> &nbsp;&nbsp;&nbsp;     %{SCHEDULER\_CONFIGURATION\_DIRECTORY}<br>     <br>     <b>Task:</b> %{SCHEDULER\_TASK\_ID}<br>     <b>Job: &nbsp; </b>%{SCHEDULER\_JOB\_NAME}&nbsp; &nbsp;&nbsp;&nbsp; <br>     %{SCHEDULER\_JOB\_TITLE}<br>     <br>     <b>Job\_chain:</b>&nbsp; %{SCHEDULER\_JOB\_CHAIN\_NAME}<br>     %{SCHEDULER\_JOB\_CHAIN\_TITLE}<br>     <br>     <b>Order</b>:&nbsp; &nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;     &nbsp;&nbsp;&nbsp; %{SCHEDULER\_ORDER\_ID}<br>     <b>in Node:</b>&nbsp; &nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;     %{SCHEDULER\_NODE\_NAME}<br>     <b>next Node:</b> &nbsp;&nbsp;&nbsp; %{SCHEDULER\_NEXT\_NODE\_NAME}<br>     <br>     <font color="#999999" face="Calibri"><small><small><font size="-1"><small><small></small></small></font></small></small></font>     <div class="moz-signature"> </div>   </body> </html> "/>

               <param  name="MailOnJobStart\_content\_type" value="text/html"/>

               <param  name="MailOnError\_to" value="uwe.risse@sos-berlin.com"/>

               <param  name="MailOnError\_subject" value="An error occurred in Job: %{SCHEDULER\_JOB\_NAME}  Task: %{SCHEDULER\_TASK\_ID}"/>

               <param  name="MailOnError\_content\_type" value="text/html"/>

               <param  name="MailOnError\_body" value="<html>   <head>      <meta http-equiv="content-type" content="text/html; charset=ISO-8859-15">      <title></title>   </head>   <body text="#000000" bgcolor="#FFFFFF">     <b>Scheduler ID:</b> &nbsp; &nbsp;&nbsp; &nbsp;&nbsp;&nbsp;     %{SCHEDULER\_ID}<br>     <b>Scheduler</b>:&nbsp;&nbsp;&nbsp; &nbsp;&nbsp; &nbsp;     &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;     %{SCHEDULER\_HOST}:%{SCHEDULER\_TCP\_PORT}<br>     <b>Configuration in:</b> &nbsp;&nbsp;&nbsp;     %{SCHEDULER\_CONFIGURATION\_DIRECTORY}<br>     <br>     <b>Task:</b> %{SCHEDULER\_TASK\_ID}<br>     <b>Job: &nbsp; </b>%{SCHEDULER\_JOB\_NAME}&nbsp; &nbsp;&nbsp;&nbsp; <br>     %{SCHEDULER\_JOB\_TITLE}<br>     <br>     <b>Job\_chain:</b>&nbsp; %{SCHEDULER\_JOB\_CHAIN\_NAME}<br>     %{SCHEDULER\_JOB\_CHAIN\_TITLE}<br>     <br>     <b>Order</b>:&nbsp; &nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;     &nbsp;&nbsp;&nbsp; %{SCHEDULER\_ORDER\_ID}<br>     <b>in Node:</b>&nbsp; &nbsp;&nbsp;&nbsp; &nbsp;&nbsp;&nbsp;     %{SCHEDULER\_NODE\_NAME}<br>     <b>next Node:</b> &nbsp;&nbsp;&nbsp; %{SCHEDULER\_NEXT\_NODE\_NAME}<br>     <br>     <font color="#999999" face="Calibri"><small><small><font size="-1"><small><small></small></small></font></small></small></font>     <div class="moz-signature"> </div>   </body> </html> "/>

           </params>

           <script  language="shell">

               <![CDATA[

dir

exit 1

               ]]>

           </script>

           <monitor  name="process0" ordering="0">

               <script  language="java" java\_class="com.sos.jitl.mail.smtp.SmtpMailMonitor"/>

           </monitor>

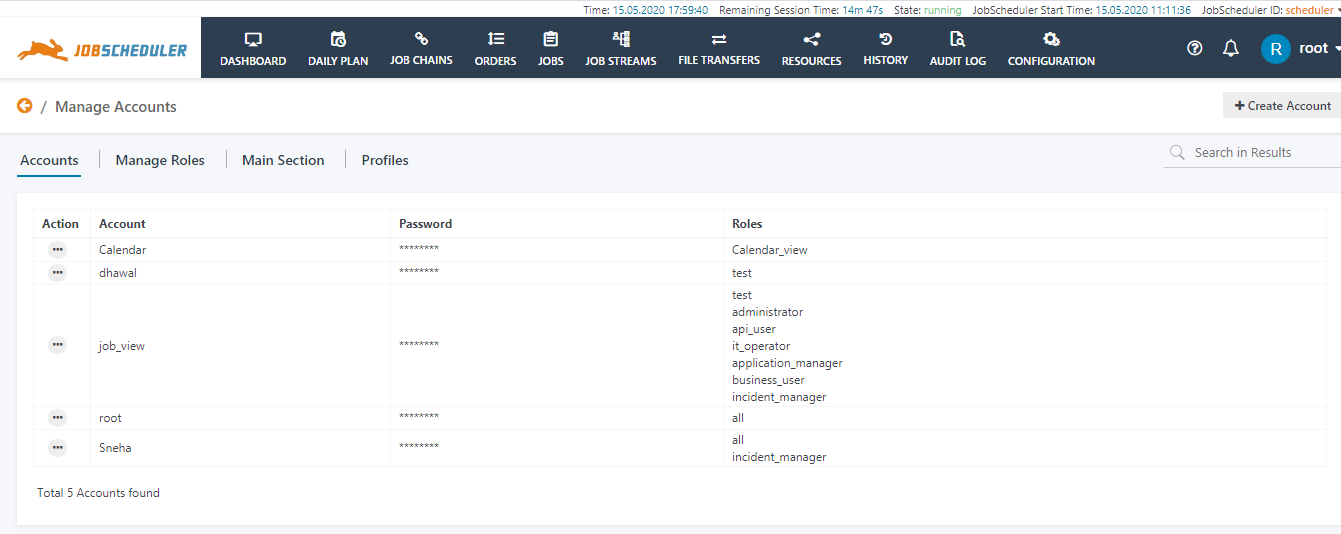
           <run\_time />

       </job>

* The above example sends an email notification to the user based on success, failure or warning.
* The `<monitor>` tag which contains the JITL mail job is used to send email to the user.
* Under the `<params>` tag notice that all the mail requirements are filled up.

1. **ROLES AND PERMISSIONS**

* Roles can be created in JOC Cockpit for that you will have to log into root user and click on the dropdown in the right corner of JOC.
* After that you will have to click on `Manage Accounts` it will redirect you to the accounts and role management section.



* You can create you custom account with a username and password and you can assign a role to it.
* Authentication can either take place in Apache Shiro or LDAP directory. Authentication against multiple realms is possible.
* Authorization is defined in roles and permissions and an example set of roles and permission is provided with JOC Cockpit installation.
* System administrators are able to define their own user roles and permissions.
* The JOC Cockpit is able to handle authentication of multiple users and their authorization for multiple JobSchedulers simultaneously.
* We can also restrict the user to view only particular jobs , unfortunately SOS berlin does not provide us with that but there is a workaround.
* The jobs which you want to restrict to particular users add that jobs to a folder. SOS Berlin provides us with the facility to restrict the visibility of the folders to the role.