|  |  |
| --- | --- |
|  |  |
| <Document type, such as Master Guide>  <Product or Solution Name - or delete this paragraph>  Document Version: 1.0 – 2012-04-18 | Confidential |
| <Main Title - (e.g. Product Name and Release)>  <Subtitle - or delete this paragraph> | |

SAPLogo.gif

|  |  |
| --- | --- |
|  |  |
| Administrator’s Guide  Integration Framework for Cloud – iPaaS Edition  Document Version: 1.0 – 2020-09-08 | Public |
| Administrator’s Guide | |

Document History

|  |  |  |
| --- | --- | --- |
| Version | Date | Change |
| 1.0 | 2020-01-04 | Release to customer (RTC) |

Table of Contents

1 Container Backend and Integration Landscape 5

2 Software Download, Installation and Image Creation 10

2.1 Prerequisites 10

2.2 Downloading and Installing the Software 12

3 Initial Landscape Setup with Tenant Shell 14

4 Administrative Tasks 15

4.1 Replacing SSL Certificates of the Reverse Proxy 15

4.2 Certificate Creation Example for the Reverse Proxy 15

4.3 Running Behind a Custom Reverse Proxy 16

4.4 Mandatory Directives for the iPaaS Tenants 17

4.4.1 Mapping URLs 17

4.4.2 Enriching Request and Response Headers 18

4.5 Restarting iPaaS Tenants 18

4.6 Restarting the Reverse Proxy 19

4.7 Removing iPaaS Tenants 19

4.8 Stopping iPaaS Tenants 20

4.9 Starting Stopped iPaaS Tenants 20

4.10 Upgrading iPaaS Tenants 21

4.10.1 Updating Images 21

4.10.2 Specifics for the iPaaS Controller Update 22

4.10.3 Updating the iPaaS Controller and the Clients 22

4.11 Creating Backups 24

4.12 Restoring Backups 24

4.13 Updating the Reverse Proxy 25

4.14 Updating the API Server 26

4.15 Installing SAP Business One DI Proxies 26

5 Advanced Administration Tasks 28

5.1 Preserving Files Across Container Restarts 28

5.2 Deriving Custom Images from the Application Image 29

5.2.1 Deriving an Image from the Base Image 29

5.2.2 Extending the Integration Framework on File System Level 29

5.2.3 Setting Custom Default Entry URLs 30

5.2.4 Extending the Database Schema and BizStore Content 31

5.2.5 Changing or Extending Default Application Settings 35

5.2.6 Creating Clients from a Custom Image 36

6 Appendix 37

6.1 Command Line User Interfaces 37

6.1.1 tenantShell Commands 37

6.1.2 Creating User Accounts for the REST API Server 39

6.2 Runtime Variables in tenantShell 39

6.3 iPaaS Controller and iPaaS Client-Specific Environment Variables 41

6.4 tenantShell REST API 41

6.4.1 Starting and Accessing the Server 42

6.4.2 Stopping the Server 42

6.4.3 Logging 42

6.4.4 Getting and Renewing an API Access Token 43

6.4.5 API Endpoints and Methods 43

# Container Backend and Integration Landscape

The installation option for cloud enables the integration framework to run in cloud environments that use operating system level virtualization or containerization, based on Docker Swarm.

Leveraging the cloud stack technologies of common data center infrastructures, it is simple to host the integration framework in modern cloud hosting environments, such as SAP Converged Cloud, Amazon AWS, Microsoft Azure, Google Cloud Platform and so on.

The integration framework provides the B1iFCloudManager standard interface to manage integration framework instances or tenants on any underlaying cloud infrastructure.

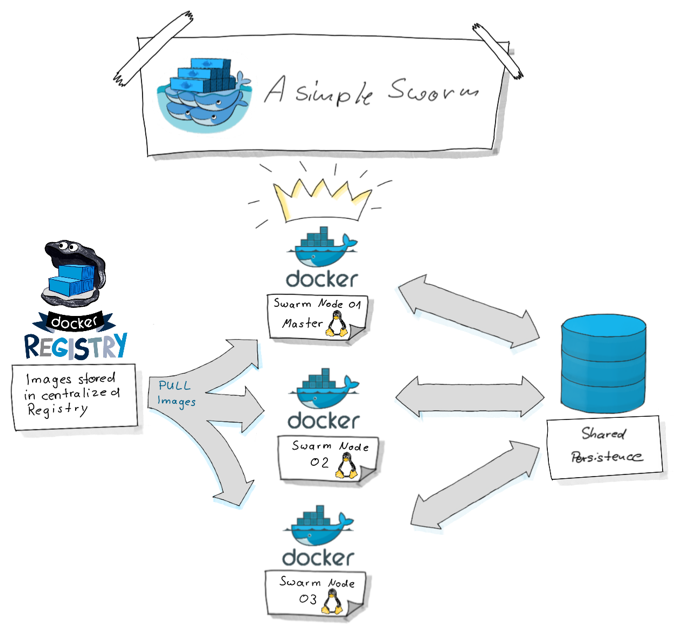
Docker Swarm

Docker Swarm is an orchestration tool for Docker containers. Using Docker Swarm, you can scale integration framework tenants over multiple nodes in a swarm. Docker Swarm allows aligning the operations infrastructure whenever business needs require it.

In addition to Docker, the industry standard containerization technology, Docker Swarm provides features like automatic scaling across entire data centers and out-of-the-box failover. However, Docker Swarm requires a slightly higher level of planning and causes higher complexity in terms of providing persistent storage for the applications.

Starting with a small environment and becoming familiar with Docker Swarm, it is possible and very straightforward to run the entire swarm on a single machine. Docker Swarm allows scaling the actual swarm by adding new machines or nodes without downtime. The feature makes it easy to scale smaller to larger setups when business requirements grow.

A Simple Docker Swarm

  
Simple Docker Swarm Setup

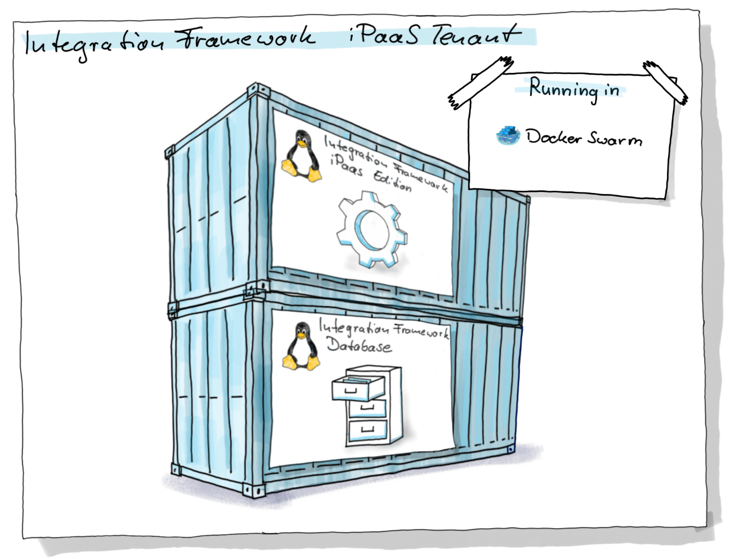
For development and test environments, you can run Docker Swarm on a single node. This setup enables you to become familiar with the technology. Once you are familiar with running the integration landscape on a single host swarm, add further nodes to your swarm. This task is straightforward and allows your swarm to become fault tolerant and scalable.

To benefit from the Docker-immanent high availability feature, a productive swarm must consist of at least two manager nodes running Linux and two worker nodes running Linux.

A highly available file server is required to host the shared persistence of the application and database containers. You can run the nodes in your data center on physical hardware, on top of your virtualization, or in public clouds such as Amazon AWS, Microsoft Azure, Google Cloud Platform and others. However, as displayed in the illustration above, a non-fault tolerant swarm can consist of two to three swarm nodes and the central file server.

Docker Swarm is very similar to Docker. In terms of managing tenants across the entire swarm, Docker Swarm feels and behaves like a simple Docker setup.

The Tenant Concept

  
iPaaS Tenant Concept

An integration framework integration Platform as a Service (iPaaS) tenant consist of two Docker containers that are logically bundled using the underlaying container orchestration:

* Integration framework iPaaS edition application (Xcellerator)
* A dedicated database server that contains the persistent data (BizStore) of the tenant.

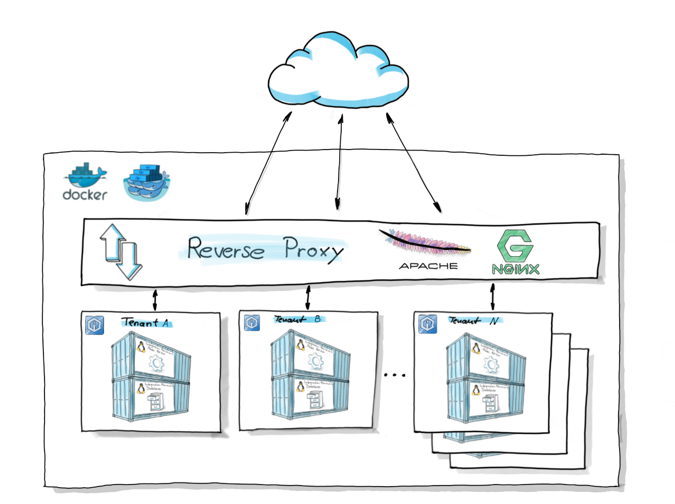
Using a container orchestration framework such as Docker Swarm, your integration landscape hosting environment becomes fault tolerant when there are enough nodes in the swarm to cover reasonable failover for all connected nodes.

Tenant Shell

tenantShell is the tenant life cycle management and orchestration tool that is part of the integration framework delivery for cloud.

tenantShell can deploy iPaaS tenants to the underlaying container orchestration. Docker Swarm as the container backend, enables tenantShell to deploy iPaaS tenants across host clusters. The hosts can be physical or virtual hosts, they can be part of a private or a public cloud. Services are easily scalable and available almost instantaneously without the need of installing any software. You can think of an iPaaS tenant deployment as an automated installation of preconfigured appliances across a virtual hosting center. tenantShell comes along with a REST API server that allows access to manage the integration landscape.

Access to iPaaS Tenants Through a Reverse Proxy

   
Access to Integration Frameworks from Outside

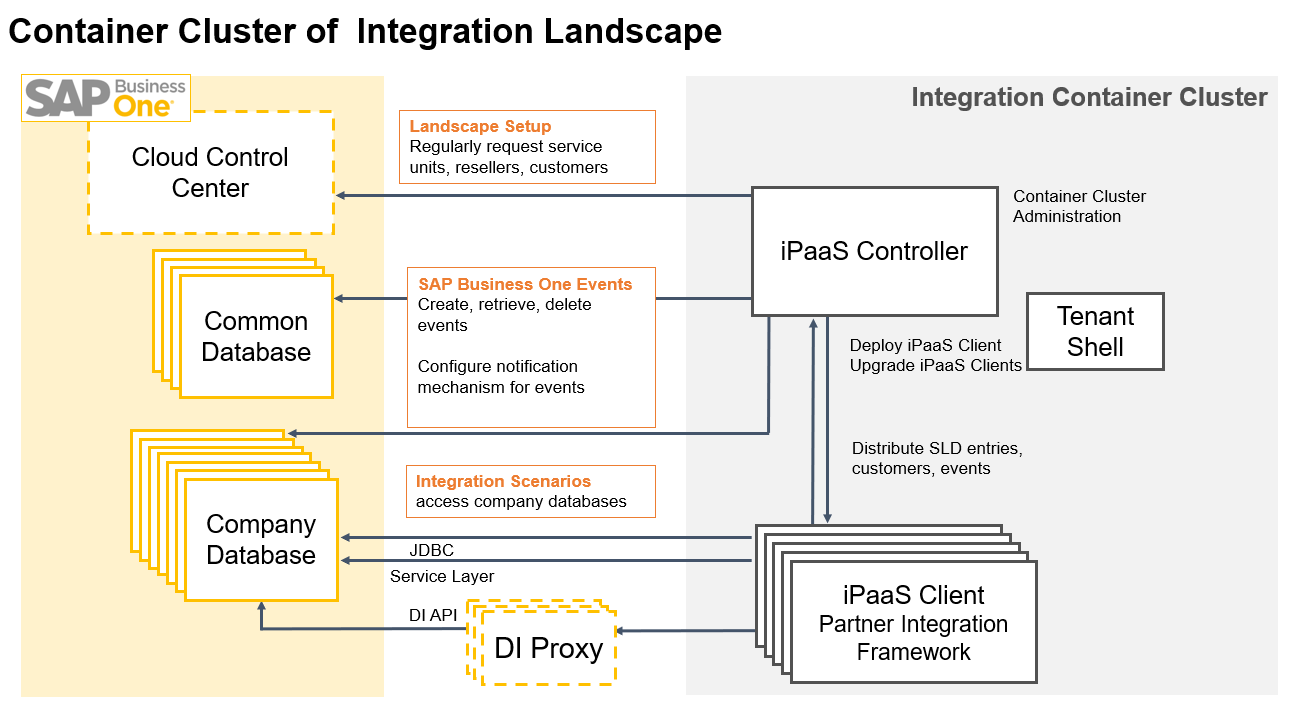
From the internet, you access individual iPaaS tenants through a reverse proxy (for example, provided by Apache with automatic configuration by tenantShell or NGINX wit manual configuration) either by using the TenantId that becomes part of the calling URL or by using distinct host and domain names (fully qualified domain names).

iPaaS Controller and iPaaS Clients

The installer deploys an iPaaS tenant as an iPaaS controller or as an iPaaS client to the integration landscape. The landscape requires exactly one iPaaS controller and as many iPaaS clients as needed. When running the initial landscape setup, the installer deploys the iPaaS controller first.

The iPaaS controller is the central component that communicates with all other components. The controller connects to the SAP Business One Cloud Control Center. From the iPaaS controller user interface, you can create iPaaS clients using the tenant shell functions. The iPaaS controller retrieves and distributes reseller, customer and service unit information to the clients. An important task of the controller is the subscription to shared resources, such as SAP Business One events, SAP ERP IDocs, and so on. The controller distributes events to subscribed clients that require the events to trigger integration scenarios.

An iPaaS client is an isolated integration framework of a partner or reseller with a dedicated database. The iPaaS client concept offers iPaaS tenancy in the integration landscape. Using the integration framework version 2 in iPaaS clients, a partner or reseller can set up integration content for many customers enabling SaaS (Software as a Service) tenancy by deployment identifier separation. A deployment identifier is available for each customer for whom a scenario package is set up. The integration scenarios access the SAP Business One company databases.

  
Integration Landscape Components

# Software Download, Installation and Image Creation

## Prerequisites

Target Audience

Although the implementation of the integration framework on Docker Swarm abstracts the complexity of the underlying operating systems and the virtualization framework, the procedures described below still require a very basic understanding of using Unix shells and a basic understanding of Docker. It is useful to know how to start simple containers. Additionally, you should be familiar with the terms image and container in the context of Docker.

Operating System

You have one of the following operating systems installed on bare metal or any type of virtualization environment:

* Debian 8 or 9 based system, such as Debian, Ubuntu or Linux Mint on x86\_64
* Suse Linux Enterprise 12 on x86\_64.

Hardware Recommendations

For test and development systems not considering any productive workloads, the minimum configuration of a virtual or physical machine should provide 2 GB of Ram and 1 CPU per client plus the same amount for the operating system.

**Example**

A system that runs three non-productive clients, should provide 4 CPU cores and 8 GB of Ram.

The scaling factor of the metrics for a productive system heavily depends on the complexity of implemented scenarios and the estimated workload. Perform some realistic scenario tests before scaling your productive hardware. Many cloud infrastructure providers allow scaling of underlying hardware even for already provisioned virtual machines.

Required Software

* Docker CE or EE from https://docs.docker.com/engine/installation/. At least version 17.
* Java Runtime Environment (JRE) 1.8
* The installer checks for some more programs to be on the PATH. The dependencies are:
* uuidgen
* sha256sum
* pwgen
* openssl
* whoami
* date
* id
* mktemp
* getfacl
* tput
* rlwarp (optional)

Most of the programs come with a default installation on Debian-based distribution. On Debian, you can run the following commands to install missing packages:

$ sudo apt-get update

$ sudo apt-get install pwgen openssl acl rlwrap openjdk-8-jre

The package manager and the list of packages required to install may vary depending on your distribution.

Required System Configuration

* On the machine running the integration framework, the 443 and 80 ports must not be in use by any program.
* The system user that runs administration tasks on the management node for the integration framework must be a member of the docker group.

Let us assume that the user name is someuser. You can choose any user name except root.

* The docker group is created during the Docker installation procedure. To check whether the user is a member of the group, perform the following statement:

$ grep docker /etc/group

The system returns:

docker:x:999:docker,someuser,someotheruser

If the user is part of the list after the last colon, everything is already set up properly. If the user is not part of the list, use the following command to add the user:

$ sudo usermod -a -G docker someuser

Running in Docker Swarm

The section is relevant, if you run more than one node in your Docker Swarm. For small test setups with a single node, you can skip the section.

* Make sure that the shared storage area exported by the file server is mounted at the same place or mount point for each Linux node.

By default, the place is at /var/B1iFCloudManager. All nodes need write permissions to the storage area.

* Using NFS (Network File System) to distribute the central storage across the swarm, do the following:
* Make sure that the file system is exported with the rw,sync,all\_squash,anonuid=<uid>,anongid=<gid>.
* Replace uid and gid with the uid and gid that is used to run the containers on all nodes.
* During the first start of tenantShell, the tool defines a directory that becomes the storage for the persistency of all tenants and their containers. Use a path under the mount point of your shared storage.

The default is /var/B1iFCloudManager/containers.

* To simplify the distribution of Docker images across the swarm nodes, we recommend using a Docker registry. All swarm nodes can pull the images from the registry server instead of using their local copies.
* It is possible to run a swarm without a central registry. Then, you must push the integration framework images to the local registry of each individual swarm node.

## Downloading and Installing the Software

The software and scripts to build Docker images are part of the integration framework for SMEs delivery. The integration framework for SMEs and the integration framework for SAP Business One are identical.

The B1if\_for\_Containers\_on\_Linux.sh installer for the integration framework for cloud is part of the installation package for Microsoft Windows.

The description below assumes that you download, install and build the images on a separate machine.

Prerequisites

The machine where you install the software and build the images, requires the following:

* The machine has a running Docker daemon in swarm mode.
* The machine has access to the internet.
* The user that installs the software must have sudo privileges.

Procedure

1. To download the installation package, go to the SAP Support Portal Software Downloads area at <https://support.sap.com/en/my-support/software-downloads.html> (login required) and click the Access downloads link in the Support Packages & Patches section.
2. Navigate to B → SAP BUSINESS ONE PRODUCTS → INTEGRATION FRAMEWORK FOR SME → INTEGRATION FRAMEWORK FOR SME 1.0 → COMPRISED SOFTWARE COMPONENT VERSIONS → INTEGRATION FRAMEWORK 1.0, select WINDOWS ON X64 64BIT version and then the B1IF10P\*.zip package, add the file to the download basket and download the file.
3. Extract the ZIP file and open the Deployments folder to obtain the B1if\_for\_Containers\_on\_Linux.sh installer.
4. Copy the B1if\_for\_Containers\_on\_Linux.sh installer to the machine with a Docker installation running in Docker Swarm.
5. To make the installer executable, enter the following:

$ chmod +x **B1if\_for\_Containers\_on\_Linux.sh**

1. Run B1if\_for\_Containers\_on\_Linux.sh.

The installer displays a help text that supports you in software installation and container creation.

1. Follow the instructions of the help text and instruct the setup procedure to create the Docker images automatically.

To trigger the image-build procedure manually, execute the following command:

$ /opt/sap/B1iFCloudManager/src/buildImages

You can let the setup push the images to your Docker registry of choice. For more information about pushing images, refer to the documentation of the docker push command.

The software is installed in the /opt/sap/B1iFCloudManager folder.

1. To create a user account for the inbuilt tenantShell REST API server, enter the following:

/opt/sap/B1iFCloudManager/bin/useradd

The user account is required by the iPaaS controller to manage the iPaaS clients.

1. Remember the user name and password for later use in the iPaaS controller.

Results

The Docker images to run the integration framework on Docker Swarm are stored in the registry of choice for either the local docker installation or the swarm.

tenantShell is installed and preconfigured for the first start and for setting up the iPaaS controller, the reverse proxy and the REST API server.

# Initial Landscape Setup with Tenant Shell

For the initial integration landscape setup, use the tenantShell orchestration tool. Start tenantShell and perform the following tasks:

* Define fully qualified domain names for the integration landscape
* Create the iPaaS controller
* Create the reverse proxy
* Create the tenantShell REST API server

This is the minimum setup for running the integration landscape in Docker Swarm.

Procedure

1. To start tenantShell, execute:

$ /opt/sap/B1iFCloudManager/bin/tenantShell

During the first start, tenantShell asks a few questions about your landscape configuration. The defaults fit to most cases.

1. When asked about the default public FQDN (fully qualified domain name), define the fully qualified domain name that is going to be used to access your landscape from the internet.

To define a distinct FQDN for each iPaaS client, you can overwrite the default later.

1. Set the first login password for B1iadmin for the iPaaS controller:

> set B1IADMINPASSWORD toP3ecret

1. Trigger the creation of the iPaaS controller:

> create

1. Create the tenantShell REST API server:

> create\_api\_server

1. Create and configure the reverse proxy:

> create\_proxy

> config\_proxy

After successful execution, the program outputs the URL to access the iPaaS controller from the internet.

1. To leave tenantShell, enter:

> quit

Results

The iPaaS controller runs. Log in to the iPaaS controller and configure the integration landscape in the iPaaS controller Landscape menu. The documentation for the landscape configuration is embedded in the landscape user interfaces. To display the documentation, click in Book icon.

For more information about runtime variables and tenantShell commands, refer to the appendix

# Administrative Tasks

## Replacing SSL Certificates of the Reverse Proxy

The reverse proxy as a gateway enables you to run iPaaS tenants in a cloud environment. With the reverse proxy, you make the tenants accessible from the internet hiding the inner topology of the cloud infrastructure at the same time. The reverse proxy is an entry point to the integration framework (iPaaS) clients and the controller running in the cloud.

Access to all tenants is enabled through SSL. The delivered certificates are self-signed, and you get a warning in the Web browser when accessing the tenants. To replace the default certificates of the reverse proxy, follow the procedure below.

Procedure

1. Provide certificates in PEM format.
2. Replace the default certificates created with the provisioning of the reverse proxy in the /opt/sap/B1iFCloudManager/containers/rev\_proxy/fs/etc/apache2/certs folder.
3. To set the new certificates active, restart the reverse proxy:

$ touch /opt/sap/B1iFCloudManager/containers/rev\_proxy/fs/bootstrap/var/RELOAD

## Certificate Creation Example for the Reverse Proxy

You can obtain valid server certificates for the reverse proxy, for example, from Let’sEncrypt (<https://letsencrypt.org/>), an open Certificate Authority.

The operating system, where the reverse proxy runs provides functions to create such SSL certificates that are accepted by all major Web browsers.

The services, that are offered by Let’sEncypt are a third party offering, that is not covered by any warranty, support and licensing of SAP. Please refer to the Let’sEncrypt Policy and legal Repository for any further information.

Procedure

1. To obtain a certificate, start a shell inside the reverse proxy container:

docker exec -it rev\_proxy\_rev\_proxy.xxx bash

1. Install Certbot, the Let’sEncrypt client that comes along with the operating system:

apk add certbot

1. Let Certbot create a certificate for your domain:

certbot certonly

1. Follow the instructions on the screen, when asked for:

1: Spin up a temporary webserver (standalone) or

2: Place files in webroot directory (webroot)

Select 2 and enter /var/www/localhost/htdocs as the path to the webroot.

1. After successful execution, link files from /etc/letsencrypt/live/your.domain.here/ to /etc/apache2/certs/ by keeping the names of the original files in /etc/apache2/certs/.
2. Restart Apache:

touch /bootstrap/var/RELOAD

## Running Behind a Custom Reverse Proxy

tenantShell provides functions to transparently create and configure a reverse proxy based on the underlaying containerization. We recommend using the inbuilt Apache HTTP Server proxy. However, you can replace or extend the proxy by a custom proxy.

The reverse proxy as a gateway enables you to run iPaaS tenants in a cloud environment. With the reverse proxy, you make the clients accessible from the internet hiding the inner topology of the cloud infrastructure at the same time. The reverse proxy is an entry point to the integration framework clients running in the cloud. To translate the external to the internal addressing schemas, the proxy requires knowledge about the internal topology.

The example below displays a reverse proxy configuration with two iPaaS tenants, using Apache with mod\_proxy and mod\_ssl:

<VirtualHost \*:443>

SSLEngine on

SSLCertificateFile /etc/apache2/certs/cert.pem

SSLCertificateKeyFile /etc/apache2/certs/privkey.pem

ServerName dev01.smbintegration-ondemand.com

ProxyRequests Off

# Name: smbhub\_8082

# Path: /opt/sap/smbHub/bin/../containers/smbhub\_8082

# Port: 8082

# TenantId: beb71ddc9805

# TenantDnsName: dev01.smbintegration-ondemand.com

<LocationMatch "/beb71ddc9805/">

ProxyPass http://172.17.0.1:8082/

ProxyPassReverse http://172.17.0.1:8082/

ProxyPassReverseCookiePath / /beb71ddc9805/B1iXcellerator

Header add X-Forwarded-SMBHub-URLPrefix "beb71ddc9805"

RequestHeader set X-Forwarded-SMBHub-URLPrefix "beb71ddc9805"

RequestHeader set X-Forwarded-SMBHub-Protocol "https"

Header add X-Forwarded-SMBHub-Protocol "https"

</LocationMatch>

# Name: smbhub\_8083

# Path: /opt/sap/smbHub/bin/../containers/smbhub\_8083

# Port: 8083

# TenantId: w6df984abe38

# TenantDnsName: dev01.smbintegration-ondemand.com

<LocationMatch "/ w6df984abe38/">

ProxyPass http://172.17.0.1:8083/

ProxyPassReverse http://172.17.0.1:8083/

ProxyPassReverseCookiePath / /w6df984abe38/B1iXcellerator

Header add X-Forwarded-SMBHub-URLPrefix "w6df984abe38"

RequestHeader set X-Forwarded-SMBHub-URLPrefix "w6df984abe38"

RequestHeader set X-Forwarded-SMBHub-Protocol "https"

Header add X-Forwarded-SMBHub-Protocol "https"

</LocationMatch>

</VirtualHost>

## Mandatory Directives for the iPaaS Tenants

To operate the iPaaS tenants, the following functions must be implemented by the reverse proxy. The example uses an Apache-specific configuration. Adjust the configuration accordingly to specifications of other proxy servers.

### Mapping URLs

The iPaaS tenants require URL mapping between internal and external URLs. SSL handling is not performed on tenant level. The reverse proxy performs the SSL handling. Therefore, the https schema is translated to http. http is only used internally, https is only used externally.

Procedure

The following statements are responsible for the mapping of external to internal URLs.

ProxyPass http://172.17.0.1:8083/

ProxyPassReverse http://172.17.0.1:8083/

Example

The external URL https://dev01.smbintegration-ondemand.com/w6df984abe38 is mapped to tenant http://172.17.0.1:8083/. Externally, you can identify a tenant by the unique TENANT\_ID (w6df984abe38).

In the example, the external ID maps to an instance of the tenant running on 172.17.0.1 on port 8083.

### Enriching Request and Response Headers

Procedure

Hand over parameters related to the cloud infrastructure and required by the integration framework server or client in an extended HTTP header

|  |  |
| --- | --- |
| Directive | Description |
| RequestHeader set X-Forwarded-SMBHub-URLPrefix | Informs the integration framework server about the TENANT\_ID |
| RequestHeader set X-Forwarded-SMBHub-Protocol | Information the integration framework server about the externally used addressing schema. Possible values are http and https. The parameter value must match the addressing schema that the proxy exposes to the internet. |
| Header add X-Forwarded-SMBHub-URLPrefix | Information the integration framework client about the TENANT\_ID that the server runs |
| Header add X-Forwarded-SMBHub-Protocol | Informs the integration framework client about the addressing schema used by the server. |
| ProxyPassReverseCookiePath | Rewrites the path string in the Set-Cookie header sent by the server, to match the externally used URL |

## Restarting iPaaS Tenants

You can restart iPaaS tenants from the system shell.

Prerequisites

You are logged on to the operating system shell that hosts the clients with the user that runs the administration tasks of the integration framework. Detect the client name of the client that you want to restart.

Procedure

1. To identify the client that you want to restart, issue the ls command in tenantShell.

The command returns a list that links client names to external URLs. The list allows you to identify the client.

Valid URLs to access the clients are:

Tenant smbhub\_8081

https://localhost/00f36d566420/

http://localhost/00f36d566420/

Tenant smbhub\_8082

https://localhost/1a215c5065ea/

http://localhost/1a215c5065ea/

Tenant smbhub\_8085

https://localhost/1a215c506fde/

http://localhost/1a215c506fde/

1. To trigger the restart for an instance, touch the reload token in the client file system in the following way, for example:

touch [path]/smbhub\_8082/app/fs/bootstrap/var/RELOAD

Replace smbhub\_8082 with the client name you want to restart and [path] with the path defined for the container persistence during the first start of tenantShell.

The restart takes up to 30 seconds.

## Restarting the Reverse Proxy

After applying changes to the reverse proxy configuration, restart the proxy. Such changes are mostly related to certificates. When running in local Docker mode, you can trigger the restart issuing the config\_proxy command in tenantShell.

Procedure

To trigger the restart of the reverse proxy without invoking tenantShell, touch the reload token of the proxy in the following way:

$ touch /opt/sap/B1iFCloudManager/containers/rev\_proxy/fs/bootstrap/var/RELOAD

The restart takes up to 30 seconds.

## Removing iPaaS Tenants

To completely remove an iPaaS tenant, tenantShell provides the rm command.

The command stops the containers of the iPaaS tenant and removes the persistent storage from the disk. You cannot restart a removed iPaaS tenant by issuing the start command, because the tenant no longer exists.

Prior to removing the tenant, the rm command creates a backup of the tenant. You can restore the tenant. For more information, see section 4.11 Restoring Client Backups.

Procedure

To trigger the removal of, for example, smbhub\_8082 using tenantShell, enter the following:

> set EXTERNAL\_PORT 8082

> rm reallyreallyreallyremove

Result

The iPaaS tenant is removed and a backup of the iPaaS tenant is available on disk.

Open the iPaaS controller, choose Landscape → iPaaS Clients, synchronize with tenant shell and click the Delete button for the removed iPaaS client.

## Stopping iPaaS Tenants

To stop an iPaaS tenant while keeping all persistent memory on disk, tenantShell provides the stop command.

Procedure

To trigger the stop, for example of, smbhub\_8082 using tenantShell, enter the following:

> set EXTERNAL\_PORT 8082

> stop

Result

tenantShell stops the containers responsible for running the iPaaS tenant and removes them from the Docker Swarm. The persistence of the iPaaS tenant still exists on disk allowing to restart the iPaaS tenant again.

## Starting Stopped iPaaS Tenants

To start an iPaaS Tenant, that was stopped using the tenantShell stop command, tenantShell provides the start command.

Procedure

To trigger a start, for example, of smbhub\_8082 using tenantShell, enter the following:

> set EXTERNAL\_PORT 8082

> start

Result

tenantShell starts the containers responsible for running the iPaaS tenant based on the same configuration and persistence the containers used before they were stopped.

## Upgrading iPaaS Tenants

If the SAP Support Portal Software Downloads area offers a new patch, a new B1if\_for\_Containers\_on\_Linux.sh is part of the download. Install the new version and build images as described in the second section of the guide and make them available in your repository.

### Updating Images

Procedure

1. During client update, the related images are pulled automatically if they are not yet present on the target machine that Docker Swarm addresses to run the client.

You can define the source of the images during the update procedure in tenantShell.

1. For your convenience, a list of default sources can be specified in the files

/var/B1iFCloudManager/etc/images.app.list

/var/B1iFCloudManager/etc/images.db.list

/var/B1iFCloudManager/etc/images.tenantshell.list

/var/B1iFCloudManager/etc/images.proxy.list

Each file contains a list of 1 to n possible image sources in the same notation as used by the docker pull command. The entry with the highest alphanumeric order is used in tenantShell as the default location for the image.

**Example for a simple list file:**

# current default Image from remote registry

my.registry.server.local/b1i\_app\_ipaas:9.3.10.17

# local version used for test systems

b1i\_app\_ipaas:9.3.10.15

1. Make sure that the sources point to valid images and that all swarm nodes have access to the referenced remote registry, if you use a remote registry. If the sources do not point to valid images, the client does not start after the update.

If there is something wrong, you can roll back by reverting to the latest backup of the client. The update procedure automatically creates such a backup.

1. Instead of assigning a specific version (tag in docker terminology), you can reference all your iPaaS tenants to the “latest” tag.

In tenantShell, call set DOCKER\_IMAGE\_TAG latest prior to calling create.

You can also define a reference to the latest in the list files mentioned above. For /var/B1iFCloudManager/etc/images.app.list, such an entry looks like the following:

# latest Image from remote registry

my.registry.server.local/b1i\_app\_ipaas:latest

### Specifics for the iPaaS Controller Update

You can independently update all iPaaS clients, the reverse proxy and the API server.

The iPaaS controller is an exception, because the iPaaS controller is the owner of the swarm internal network that is shared by all components. Additionally, the iPaaS controller is also the owner of a set of secrets that is shared between the controller and the clients. You can only stop and update the controller without obtaining certain warnings from Docker, if all other components are stopped.

Usually, you can ignore the warnings from the Docker daemon, because Docker’s internal resource management correctly handles the life cycle of the network and secrets and reattaches everything to the iPaaS controller once it is restarted.

However, this is a Docker internal functionality that SAP does not control. Note that future releases of Docker may behave differently in terms of resource life cycle management. An update of the iPaaS controller that works independently from the life cycle management for shared resource in Docker requires all iPaaS clients, the reverse proxy and the API server to be stopped before you can update the iPaaS controller.

### Updating the iPaaS Controller and the Clients

During the update procedure, all containers (iPaaS controller, iPaaS client, reverse proxy and API server) can be shifted to a specific version of the related images. The version is usually the latest version. However, the version can be any other version.

#### Running Containers on Latest Version

If you want to run your containers on the latest version, it is enough to stop and start the containers.

Prerequisites

* A new version of the integration framework Docker images is available either in all local registries or in a remote registry either under the “latest” tag.
* For updating the iPaaS controller in a safe way, you have stopped all iPaaS clients, the proxy and the API server.

Procedure

1. To trigger the update while explicitly setting the image sources, execute the following commands inside tenantShell or save them to a script and pipe the script into tenantShell:

set EXTERNAL\_PORT 8082

stop

start

quit

1. Replace 8082 with the numbering part of the tenant name returned by the ls command in tenantShell.

Results

You have the following available:

* An updated tenant derived from the latest images

#### Running Containers on a Specific Version

If you want to run your containers on a specific version, use the tenantShell update command. The command can set the image names and tags. Additionally, the update command automatically stops and then restarts the tenants during upgrade and writes a backup of the old tenant.

Prerequisites

* A new version of the integration framework Docker images is available either in all local registries or in a remote registry either under an explicit new version(tag) or referenced by the “latest” tag.
* For updating the iPaaS controller in a safe way, you have stopped all iPaaS clients, the proxy and the API server.

Procedure

1. To trigger the update while explicitly setting the image sources, execute the following commands inside tenantShell or save them to a script and pipe the script into tenantShell:

set EXTERNAL\_PORT 8082

set DOCKER\_IMAGE\_APP my.registry.server.local/b1i\_app\_ipaas

set DOCKER\_IMAGE\_DB my.registry.server.local/b1i\_db\_ipaas

set DOCKER\_IMAGE\_TAG 9.3.10.17

update

quit

The latest tag can also be set explicitly in this procedure. Use ‘set DOCKER\_IMAGE\_TAG latest’ instead of ‘set DOCKER\_IMAGE\_TAG 9.3.10.17’ if you have chosen to always use the latest image version.

1. Replace 8082 with the numbering part of the tenant name returned by the ls command in tenantShell.
2. Replace the image sources according to your registry by omitting the tag.
3. Explicitly set the tag for both images by changing 9.3.10.17 or latest according to your image source.

You cannot use different tags for the two images.

1. You can omit the set DOCKER\_IMAGE\_X commands, if you want to use the default images sources as defined in the section above.

If you have not defined any default image sources, the set DOCKER\_IMAGE\_X commands are mandatory.

Results

You have the following available:

* An updated tenant derived from the latest images
* An automatically created backup version of the same tenant

The updated and the backup tenant rely on the same persistent storage. Do not run them simultaneously.

## Creating Backups

You can back up the iPaaS controller and iPaaS clients using tenantShell. Backups will be saved to /var/B1iFCloudManager/backups/. The backup contains the tenant including the status of the application at the point the backup was triggered. A backup is a snapshot of the entire application. You can bring the backup to a different machine. If the second machine runs the same application, you can restore the backup and continue running the application from the point the backup was created.

For more information about restoring backups, see section 4.11 Restoring Client Backups

Procedure

To trigger a backup of, for example, smbhub\_8082 using tenantShell, enter the following:

> set EXTERNAL\_PORT 8082

> save

Results

During backup, the client has a downtime of a few seconds. The downtime is related to the machine performance. The downtime should not exceed one minute.

To be able to restore a backup later, do not change the directory names in /var/B1iFCloudManager/backups/. You can copy or move valid backups to and from the folder using general file operations.

## Restoring Backups

Prerequisites

Tenants that you want to restore to a host must not be available prior to restoring them. It is not possible to overwrite existing tenants. The unique identifier for a tenant is the name that is part of the backup name as the leading part until (not including) the first “-”.

Procedure

1. To list backups available in the backup folder using tenantShell, enter the following:

> ls\_backups

The program returns, for example:

smbhub\_8088-2018-01-05-14-06-21

smbhub\_8088-2018-01-05-14-11-22

smbhub\_8082-2018-01-05-14-22-23

1. To restore a backup using tenantShell, enter the following:

> restore smbhub\_8088-2018-01-05-14-06-21

tenantShell requires the backup name handed over as an argument to the restore command.

## Updating the Reverse Proxy

The reverse proxy can be shifted to a new version of the reverse proxy image once it is available. You can choose whether you want to use a specific version of the image or the latest one.

Prerequisite

A new version of the reverse proxy Docker image is available either in all local registries or in a remote registry either under an explicit new version(tag) or referenced by the “latest” tag.

Procedure

To trigger the update, execute the following commands inside tenantShell or save them to a script and pipe the script into tenantShell:

set DOCKER\_IMAGE\_TAG 9.3.10.17

stop\_proxy

start\_proxy

quit

Instead of using a specific image version by using, for example, ‘set DOCKER\_IMAGE\_TAG 9.3.10.17’, you can also specify the latest tag by using ‘set DOCKER\_IMAGE\_TAG latest’ instead of ‘. If You omit the set DOCKER\_IMAGE\_TAG command, tenantShell will automatically assign the default as defined in the image sources. You can check which one it is, by issuing the show command in tenantShell.

Results

An updated reverse proxy is available that is derived from the latest or a specific image.

## Updating the API Server

The API server can be shifted to a new version of the reverse proxy image. You can choose whether you want to use a specific version of the image or the latest version.

Prerequisite

A new version of the API server Docker image is available either in all local registries or in a remote registry either under an explicit new version(tag) or referenced by the “latest” tag.

Procedure

To trigger the update, execute the following commands inside tenantShell or save them to a script and pipe the script into tenantShell:

set DOCKER\_IMAGE\_TAG 9.3.10.17

rm\_api\_server reallyreallyreallyremove

create\_api\_server

quit

Instead of using a specific image version by using for example ‘set DOCKER\_IMAGE\_TAG 9.3.10.17’, you can also specify the latest tag by using ‘set DOCKER\_IMAGE\_TAG latest’ instead of ‘. If You omit the set DOCKER\_IMAGE\_TAG command, tenantShell will automatically assign the default as defined in the image sources. You can check which one it is, by issuing the show command in tenantShell.

Results

An updated API server is available that is derived from the latest or a specific image.

## Installing SAP Business One DI Proxies

If integration scenarios access SAP Business One through the DI API, one or more DI Proxies must be available. Obtain the DI Proxy from the SAP Business One Cloud installation. Alternatively, the integration framework for SMEs installation package also contains an installation program for DI Proxies.

Prerequisites

The DI Proxy version and the DI API adapter version must match.

Procedure

1. From the installation package, extract the B1Cloud\_DIProxy\_Installer.zip package in the Deployments folder.
2. Extract the zip file, go to the Technology folder and run setup.exe.

In the Choose Installation Components step, the SAP Business One DI Proxy is already selected.

1. Click Next and choose the Windows machine (presentation server) and location for the DI Proxy installation.
2. Click Next and then Install.

# Advanced Administration Tasks

The following sections are only relevant, if you want to enhance the integration framework default software.

## Preserving Files Across Container Restarts

Apart from the database content, an integration framework instance has no persistent storage by default. However, some directories are configured to be persistent, because they contain applications logs. The logs can be of interest independent of the life cycle of a container or application instance. Everything not included in persistent directories is no longer available after client or container restarts.

To overcome this limitation, configuration settings are available that let you keep additional files persistent across a container restart.

Procedure

1. To keep files persistent across container restarts, add the files to the /opt/tomcat/webapps/B1iXcellerator/preserve.list file inside the respective application container.

The file contains the following files by default:

* ../../bin/setenv.sh
* xcellerator.cfg
* urlshortcuts.properties
* index.htm
* preserve.list

1. You can add more files at runtime, if required. All changes take immediate effect. When changing the file, consider the following rules:

* All paths must be relative to the webapp root directory.
* Add one file per line.
* File names (without path) must be unique.
* Empty lines are omitted
* Non-existing files are omitted
* Directories are omitted

1. You can make changes to /opt/tomcat/webapps/B1iXcellerator/preserve.list inside a shell in the container. To gain access, you can exec into the container. Issue a docker exec command on the host that runs tenantShell.

$ docker exec -it <container name> bash

For more information, see the documentation of docker exec command**.**

## Deriving Custom Images from the Application Image

The default integration framework application images provide an interface to incorporate custom functions. The interface allows you to do the following:

* Extend the integration framework on file system level
* Automatically import custom content into the BizStore
* Extend the database schema
* Change or extend the default application settings

General Prerequisites

* For the following sections, you need a basic understanding of the container and image terms in the Docker context.
* You need basic knowledge about the structure, syntax and functionality of Dockerfiles and the docker build command.

### Deriving an Image from the Base Image

By default, the B1iFCloudManager general interface ships an image that is tagged as b1i\_app\_ipass:lastest. The image contains the B1iXcellerator Web application and is preconfigured for a first run in the Apache Tomcat servlet container. You can derive custom images from this image using the default Docker functions allowing to derive images from each other.

Procedure

1. Provide a Dockerfile that references the desired base image.

For the integration framework, the base image is tagged as b1i\_app\_ipass:lastest. The image must be accessible by the local Docker installation that is used to create the custom image.

A very basic Dockerfile that derives from the integration framework application image looks as follows:

FROM b1i\_app\_ipaas:latest

RUN echo "deriving from b1i\_app\_ipaas:latest"

1. A docker build -t custom\_b1i\_app issued in the same directory where the Dockerfile resides creates a new image tagged as custom\_b1i\_app:latest and outputs the string "deriving from b1i\_app\_ipaas:latest" during the build run of the new image.

### Extending the Integration Framework on File System Level

On file system level, you can perform extensions to the integration framework application image.

Procedure

1. Use the COPY command in Dockerfile to copy any type of file-based content to the /bootstrap/ext directory in the image.

Any file or directory structure that you copy here at build time of the image, is automatically inserted into the directory structure of the B1iXcellerator Web application. Note that the operation overwrites any existing file in the base image.

1. To use the function, create a directory called ext/app in the same directory where the Dockerfile resides. The app directory under ext can contain any kind of subsequent directory structure that extends or substitutes existing parts of the B1iXcellerator Web application.
2. To copy the directory to the new image that you create, add the following line to the Dockerfile:

COPY ext /bootstrap/ext/

### Setting Custom Default Entry URLs

Procedure

1. Make sure that the current integration framework application image is accessible by your Docker daemon.

The docker images command should at least return the following information:

REPOSITORY TAG IMAGE ID CREATED SIZE

b1i\_app\_ipaas 9.3.8.09 fb5606634a64 3 hours ago ...

b1i\_db\_ipaas 9.3.8.09 dfa4533adff79 3 hours ago ...

1. Create a Dockerfile with the following content:

FROM b1i\_app\_ipaas:9.3.8.09

COPY bootstrap /bootstrap/

1. Create the bootstrap/bin directory.
2. In the directory, create the urls.inc file with the following content:

XCL\_START\_URL\_CONTROLLER='exec\/dummy\/com.sap.b1ip.system.cc\/bfd\/AdminConsole.bfd?!defdoc=\/com.sap.b1i.common\/menu\/paasctrl.xml'

XCL\_START\_URL\_CLIENT='exec\/dummy\/com.sap.b1ip.system.cc\/bfd\/AdminConsole.bfd?!defdoc=\/com.sap.b1i.common\/menu\/paas.xml'

1. Set the XCL\_START\_URL\_CONTROLLER and XCL\_START\_URL\_CLIENT variables to the start URL that you want to use for redirection when accessing the integration framework.

Escape each slash ‘/’ with a backslash '\'.

1. Build the custom docker image:

docker build -t testimage

1. Tag the new image according to the base image.

docker tag testimage:lastest testimage:9.3.8.09

1. Start tenantShell.
2. Configure tenantShell to use the custom image:

set DOCKER\_IMAGE\_APP testimage

1. Configure tenantShell to use the correct version of your images. The database image must have the same tag:

set DOCKER\_IMAGE\_TAG 9.3.8.09

1. Create a new client:

create

Results

* After configuring the reverse proxy, you can access the new client.
* After successful login, you are redirected to the new URL

### Extending the Database Schema and BizStore Content

The integration framework provides an interface that allows custom imports into the BizStore, extending the default database schema and executing BizFlows.

You can provide extension data to the interface using the dbinit directory in the root directory of the B1iXcellerator Web application. The integration framework processes files that are in the directory at start time according to a set of special rules.

You can provide extension data to the interface by placing the data under dbinit using the extension interface described in the previous section. Provide all text-based files in UTF-8.

Procedure

1. To provide extension data to the interface, create the ext/app/dbinit directory in the directory, where the Dockerfile resides.
2. Add the COPY ext /bootstrap/ext/ line to the Dockerfile.

Any data in the ext/app/dbinit directory is injected into the integration framework during image build time.

Results

Later at application start time, the data is processed during the instantiation phase of any client that is run from this image.

#### Defining Naming Schemes and Order of Processing

Make sure that there are no naming conflicts with the default content when adding content to dbinit.

All files in the dbinit directory are processed in alphabetical order. Based on a key that consists of the dataset and version of a file, the integration framework ensures that the processing of a file happens exactly once.

Procedure

1. Investigate the dbinit directory of the target version of the integration framework to ensure that there are no naming conflicts with your custom naming scheme.
2. Build file names in the dbinit directory using the following list of components:

|  |  |
| --- | --- |
| Component | Description |
| numeric order | Three-digit order number. According to its position in the file names, this is the main criterion for the order of processing. |
| bizstore dataset | Target dataset in the BizStore |
| database type | Three-character identification string for the database system  The string must correspond to the property xcl.dbinitdb.{xxx}=”{product name}” in xcellerator.cfg.  The following ID strings are available for RDBMS names:   * msy: MySQL or MariaDB * ora: Oracle * db2: DB2 * mdb: MaxDB * han: SAP HANA * sql: Microsoft SQL * der: Apache Derby |
| major version.minor version.patch level | Numeric version string as “major.minor.patch” |
| platform | ID for the platform to process a file on:   * all: all platforms * win32: Windows 32 bit * win64: Windows 64 bit * lin: Linux on X86\_64 * xpi: Linux on ARM 32 |
| execution policy | Possible values:   * c: Continue processing dbinit on error * s: Stop processing dbinit on error |

#### Processing Rules for dbinit

The processing rules for extension data in the dbinit directory are based on a special naming convention for the provided files and on the file versioning.

* The versioning is part of the naming scheme and guarantees that extension data entities are processed exactly once.
* All files are processed in strict alphabetical order.
* There is support for different types of extension data. Each type serves a special purpose.
* To process any extension data, make sure that the xcl.dbinit property is set to true in xcellerator.cfg.

#### Extension Data Types and Their Purpose

Zip Files and Directories

The integration framework uses zip files and directories to import files into the data structure of the BizStore. The naming of zip files must follow the naming convention:

{numeric order}.{bizstore dataset}.{major version}.{minor version}.{patch level}.{platform}.{execution policy}.zip

The content of zip-files must follow the usual dataset/group structure of the BizStore.

Example

com.sap.b1ip.system.cc

xsl

file1.xsl

file2.xsl

xsd

file1.xsd

file2.xsd

…

…

com.sap.b1ip.system.xc

xsl

file1.xsl

file2.xsl

…

xsd

file1.xsd

file2.xsd

…

…

Directories do not require the dataset level in their content because their target dataset is taken from the name of the directory. The naming scheme for directories is the following:

{numeric order}.{bizstore dataset}.{major version}.{minor version}.{patch level}.{platform}.{execution policy}.dir

A directory contains the group structure directly in the root:

xsl

file1.xsl

file2.xsl

xsd

file1.xsd

file2.xsd

…

SQL Scripts

There are two types for SQL scripts:

* The script type with the sql file extension is used as a database definition script. The script is executed according to the execution policy.
* The script type with the sqr extension serves as a rollback script for a sql script with the same base name. Such a rollback script is always optional.

If the rollback script exists, it is processed, if the sql script with the same base name fails to execute successfully.

Provide the files in UTF-8. The naming convention for sql and sqr scripts must have the following pattern:

{numeric order}.{database type}.{major version}.{minor version}.{patch level}.{platform}.{execution policy}.{sql|sqr}

Each script can contain any number of distinct SQL instructions. The first line of an SQL script must be a select statement.

The select statement is in place as a test condition for the execution of all subsequent instructions in one file. All subsequent SQL instructions are processed, if the test condition returns an empty record set.

The example displays a script including the test condition:

Example

SELECT id FROM sysobjects WHERE id = object\_id(N'[dbo].[KEYMAP]')

CREATE TABLE %DBOWNER%.KEYMAP (SYSID nvarchar(10), LKEY nvarchar(50), GKEY nvarchar(50))

CREATE INDEX KEYMAP1 ON %DBOWNER%.KEYMAP (SYSID,LKEY)

The following special strings are superseded with runtime values at execution time:

%DBOWNER%

%DBUSER%

%JDBCURL%

BizFlow Processing Command Files

BizFlow processing command files allow the execution of BizFlows at integration framework start time. The naming convention is the following :

{Numeric order}.{database type}.{major version}.{minor version}.{patch level}.{platform}.{execution policy}.bpc

The files can contain any number of BizFlows that are executed according to the provided execution policy. Each line defines one BizFlow execution as:

{BizFlow URI} [{properties list}]

The BizFlow URI can be absolute or relative. If you use a relative URI, it is extended with the dataset provided in the file name. The property list is optional. The list is comma-separated with 1..n properties.

The following default properties are always handed over to the executed BizFlows at runtime:

dbinit.dataset

dbinit.vold

dbinit.vnew

dbinit.platform

dbinit.jdbcurl

### Changing or Extending Default Application Settings

To change the xcellerator.cfg, default settings, the base image provides an extension interface through the /bootstrap/ext/conf directory.

Valid Java properties files with the.properties extension are merged with xcellerator.cfg of the integration framework Web application when the container runs for the first time. Existing xcellerator.cfg settings are substituted with settings in the properties files.

Procedure

1. To use the interface, create a directory called ext/conf in the directory where the Dockerfile resides. This directory can contain any number of \*.properties files.
2. A suitable COPY command in the Dockerfile places the properties files under /bootstrap/ext/conf of the new image.

Example

Content of Dockerfile:

FROM b1i\_app\_ipaas:latest

COPY ext /bootstrap/ext/

Directory structure in parallel to Dockerfile:

ext/conf/

ext/conf/mycustom\_settings.properties

Content of mycustom\_settings.properties:

#enable webdav

xcl.webdav=full

# reduce default log level to SEVERE

java.util.logging.ConsoleHandler.level=SEVERE

com.sap.b1i.bizprocessor.level=SEVERE

com.sap.b1i.coordservice.level=SEVERE

com.sap.b1i.utilities.level=SEVERE

com.sap.b1i.dblayer.level=SEVERE

com.sap.b1i.xcellerator.level=SEVERE

To create the new image, open a shell in the directory where Dockerfile resides and issue:

$ docker build -t my\_custom\_app\_image .

### Creating Clients from a Custom Image

Prerequisites

A custom application image is available either in the local docker registry or a central registry, if Docker runs in swarm mode.

Procedure

To use the image to instantiate the application container, issue set DOCKER\_IMAGE\_APP followed by the name of the desired image in tenantShell.

Example

> set DOCKER\_IMAGE\_APP my\_custom\_app\_image

> set DOCKER\_IMAGE\_TAG my\_tag

> create

> config\_proxy

# Appendix

## Command Line User Interfaces

The user interface and the API are implemented in two service applications saved at /opt/sap/B1iFCloudManager/bin.

|  |  |
| --- | --- |
| Program | Description |
| tenantShell | Is a shell-style management application for clients |
| useradd | Adds user account for accessing the tenantShell REST API server and sets passwords for these accounts |

### tenantShell Commands

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Command | Description | Example | Docker Swarm | REST API |
| ls | Lists clients with a basic set of parameters |  | Yes | Yes |
| show | Shows status of client-level parameters |  | Yes | No |
| set | Sets a client-level parameter | set TENANT\_ID 1a215c5065ea | Yes | No |
| create | Creates a client with current set of client-level parameters |  | Yes | Yes |
| update | Updates a client to the latest version of available images | set EXTERNAL\_PORT 8082  update | No | Yes |
| rm | Removes a client identified by EXTERNAL\_PORT  The client is deleted without further confirmation. | rm reallyreallyreallyremove | Yes | Yes |
| save | Creates a client backup. Select the client by setting the EXTERNAL\_PORT variable. | set EXTERNAL\_PORT 8082  save | No | Yes |
| restore | Restores a backup transferred from a different machine or created on the local machine.  Provide the name of the backup as an argument to restore. | restore smbhub\_8088-2018-01-05-14-06-23 | No | No |
| ls\_backups | Lists available backup names |  | No | No |
| create\_proxy | Creates proxy |  | Yes | Yes |
| config\_proxy | Configures proxy to forward traffic to and from all clients |  | Yes | Yes |
| rm\_proxy | Removes proxy.  The proxy is deleted without further confirmation | rm\_proxy reallyreallyreallyremove | No | Yes |
| stop\_proxy | Stops the proxy while keeping the configuration data |  | Yes | Yes |
| start\_proxy | Starts a stopped proxy |  | Yes | Yes |
| create\_api\_server | Creates the tenantShell API Server |  | Yes | No |
| rm\_api\_server | Removes the tenantShell API Server | rm\_api\_serverreallyreallyreallyremove | Yes | No |
| ls\_app\_images | List all images from /opt/sap/B1iFCloudManager/etc/images.app.list |  | Yes | Yes |
| ls\_db\_images | List all images from /opt/sap/B1iFCloudManager/etc/images.db.list |  | Yes | Yes |
| ls\_proxy\_images | List all images from /opt/sap/B1iFCloudManager/etc/images.proxy.list |  | Yes | Yes |
| quit | Quits the shell |  | Yes | No |
| help | Shows a list of supported commands when called without arguments. Shows command-specific help when called with a command as an argument | help set | Yes | No |

### Creating User Accounts for the REST API Server

The tenantShell REST API server is secured by an own access management. For the iPaaS Controller or any other service being able to access this API, at least one user must be created inside the server’s user database. The tool useradd is provided to create such entries in the user database.

Procedure

To create a user, execute the command:

/opt/sap/B1iFCloudManager/bin/useradd

The tool asks you for a user name, a password and the confirmation of the password. After successful creation of the user, the iPaaS controller as well, as any other REST client can use the provided credentials to access the tenantShell REST API. For more information on using that API, please refer to the section “tenantShell REST API”.

## Runtime Variables in tenantShell

|  |  |  |
| --- | --- | --- |
| **Variable** | **Example** | **Description** |
| SMBHUB\_DOMAINNAME | localhost | External DNS name to access a client from the internet or local network |
| EXTERNAL\_PORT | 8081 | Unique port number of a client.  The number internally identifies a client |
| TENANT\_ID | d5a59773022b | Unique client ID of a client  The string externally identifies a client |
| LIMIT\_CPU\_APP | 1.5 | Resource limits for containers. Memory limits are applied in gigabytes.  For more information, see docs.docker.com/engine/admin/resource\_constraints/ |
| LIMIT\_MEM\_APP | 2 |
| LIMIT\_CPU\_DB | 0.5 |
| LIMIT\_MEM\_DB | 1 |
| B1IADMINPASSWORD | toP3ecet | Initial password for B1iadmin. Do not leave the parameter empty. |
| PARTNER\_NAMESPACE | xxx | Pre-set for the namespace prefix for scenario development |
| ENABLE\_XCL\_HTTP\_XSRF | 1 | Enable or disable cross-site request forgery prevention by adding the XSRF token to the browser URL  Possible values are 0 | 1 |
| DOCKER\_IMAGE\_APP | b1i\_app\_ipaas | Name of the image used to instantiate the client application container |
| DOCKER\_IMAGE\_DB | b1i\_db\_ipaas | Name of the image used to instantiate the client database container |
| DOCKER\_IMAGE\_PROXY | proxy | Name of the image used to instantiate the reverse proxy container |
| DOCKER\_IMAGE\_API\_SERVER | b1i\_tenantshell | Name of the image used to instantiate the REST API server container |
| DOCKER\_IMAGE\_TAG | latest | Tag of the images to use for creating containers |
| FQDN\_MAPPING | 0 | Enables or disables the FQDN mapping in the reverse proxy for a client.   * When set to 0 (default), the reverse proxy forwards traffic to the client according to the tenantId. The tenantId must be part of the calling URL as first part of the path portion of the URL. * When set to 1, the reverse proxy forwards traffic to the client based on the domain name. Do not extend the URL with a tenantId. Clients using FQDN mapping do not have a tenant ID   You can mix clients with tenant ID mapping and clients using FQDN mapping in a swarm. However, domain names used for FQDN mappings must be unique across the swarm.  When creating a client with FQDN\_MAPPING set to 1, set SMBHUB\_DOMAINNAME accordingly.  Possible values are 1|0 |

## iPaaS Controller and iPaaS Client-Specific Environment Variables

When running in Docker Swarm, tenantShell automatically sets up the iPaaS controller and iPaaS clients in the swarm. The iPaaS controller is the landscape component that communicates with all other components, for example, the SAP Business One Cloud Control Center through the SLD service API, tenantShell and the iPaaS clients. iPaaS clients are isolated integration frameworks of partners.

There is exactly one iPaaS controller in the landscape. tenantShell creates the iPaaS controller as the first component in a swarm. All subsequently created components are iPaaS clients. To manage iPaaS clients from the controller, each client registers at the iPaaS controller. The registration exchanges all relevant information between the controller and a client to enable the controller to manage clients and to dispatch SAP business One events. The controller and the clients expose a role-specific management API that implements the interfaces for controller/client communication and interaction. All information required by clients to participate in the controller/client communication, is exchanged through Docker secrets and environment variables. The table displays the relevant environment variables.

|  |  |  |
| --- | --- | --- |
| Environment Variable | Definition on Client | Definition on Controller |
| SAP\_SMBHUB\_PAAS\_CONTROLLER\_NAME | Swarm internal network name of the controller | Own swarm internal network name |
| SAP\_SMBHUB\_PAAS\_CLIENT\_NAME | Own swarm internal host name | Own swarm internal host name |
| SAP\_SMBHUB\_IS\_PAAS\_CONTROLLER | Always 0 on all clients | 1 on the controller |
| SAP\_SMBHUB\_BUDDY\_TOKEN\_PATH | Absolute path to file containing the access token (name of secret in Docker Swarm) for accessing the controller | None |
| SAP\_SMBHUB\_OWN\_TOKEN\_PATH | Absolute path to file containing the client access token (name of secret in Docker Swarm). Is sent to controller at registration time. | None |
| SAP\_SMBHUB\_SALT | String added to access token (content of file SAP\_SMBHUB\_BUDDY\_TOKEN\_PATH) before building the SHA256 hash | String added to access token (content of file SAP\_SMBHUB\_OWN\_TOKEN\_HASH\_PATH) before building the SHA256 hash |
| SAP\_SMBHUB\_OWN\_TOKEN\_HASH\_PATH | Absolute file system path to the file that contains the SHA256 hash of tenant access token + SAP\_SMBHUB\_SALT | Absolute file system path to the file that contains the SHA256 hash of client access token + SAP\_SMBHUB\_SALT |

## tenantShell REST API

Integration framework for cloud offers a dedicated server allowing access to certain functions of the tenantShell command line interface through a REST API. The following steps are required to make use of the functions.

### Starting and Accessing the Server

Procedure

1. To start the server, perform the following statement in tenantShell:

> create\_api\_server

1. Create at least one user for the server:

/opt/sap/B1iFCloudManager/bin/useradd

1. Point a RESTful client to https://[external domain name]/admin/[endpoint].

For more information about how to start the reverse proxy, refer to section 6.1 [Setting Up the Inbuilt Proxy](#_Setting_Up_the).

The endpoint names follow the naming of related commands in tenantShell.

You can call the endpoints with or without a pending /. The following URLs are valid:

https://[external domain name]/admin/[endpoint]

https://[external domain name]/admin/[endpoint]/**.**

Note: By default, the REST API is not exported via the reverse proxy and can only be accessed in the swarm. The swarm internal host name of the API server container is tenantshell. Internal to the swarm, the REST API can be accessed by http://tenantshell:8888/[endpoint]/**.**

To enable external access to the REST API, the property EXPOSE\_REST\_API in /var/B1iFCloudManager/etc/properties must be set to 1 followed by a call of config\_proxy in tenantShell.

1. When connecting for the first time, access the token endpoint to obtain a valid access token.
2. Add the token to the HTTP header x-api-key field for any subsequent call to other endpoints.

For more information, see section 9.4 Getting and Renewing a Token for API Access

### Stopping the Server

Procedure

To stop the server, perform the following action in tenantShell:

> rm\_api\_server

### Logging

The REST API writes logs to stdout and into /opt/sap/B1iFCloudManager/log/controlside.log.0 by default. Logs can also be retrieved using docker logs [container name] command.

Procedure

1. To configure logging for the REST server, configure the /opt/sap/B1iFCloudManager/etc/server.properties file.

The first start of the REST server creates and enriches the file. The file is not available after installation and prior to the first start of the server via tenantShell command create\_api\_server.

1. The following default log configuration is available. You can change the settings according to the documentation of java.util.logging.

.level=ALL

Handlers=java.util.logging.FileHandler, java.util.logging.ConsoleHandler

java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter

java.util.logging.ConsoleHandler.level=ALL

java.util.logging.FileHandler.pattern=/opt/sap/B1iFCloudManager/log/controlside.log.%g

java.util.logging.FileHandler.level=INFO

### Getting and Renewing an API Access Token

Procedure

1. Make the first connection attempt against the token endpoint.

The endpoint uses valid user credentials as input and returns an API token. For the first connection, you can either use GET or POST to send the credentials.

* The GET request requires the credentials as basic authentication according to RFC 2617.
* The POST request requires the credentials sent as the message payload.

1. For any subsequent call to other endpoints, ensure that the token is part of the HTTP x-api-key header field.
2. The API tokens are valid until a server restart. However, the client does not know when the invalidation happens. Make sure that the client can recover from a 401 response by obtaining a new token. Then, the client must call the token endpoint again.

### API Endpoints and Methods

#### token

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Creates a new API token and returns the token to the client |
| Request | {  "userName": "username", "password": "password"  } |
| Response | {  "token": "api token"  } |

|  |  |
| --- | --- |
| Method | Description |
| **DELETE** | Invalidates an API token |
| Request | Not required |
| Response | Not required |

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Creates a new API token and returns it to the client. Send access credentials following the basic authentication scheme |
| Request | Not required |
| Response | {  "token": "api token"  } |

#### ls

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Lists clients with a basic set of parameters |
| Request | Not required |
| Response | {  "tenants": [  {  "internalId": "internal tenant ID",  "hostname": "swarm internal hostname of container",  "dbImageName": "name of db image",  "appImageName": "name of app image",  "httpUrl": "http://[external dns name]/[tenantId}/",  "name": "swarm internal name of application container",  "tenantId": "external ID of tenant",  "httpsUrl": "https://[external dns name]/[tenantID}/",  "imageTag": "tag of both images",  "status": "[running|stopped]"  },  …  ],  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### create

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Creates a client. After successfully calling the method, call config\_proxy |
| Request | {  "resellerId": "[Reseller ID]",  "partnerNameSpace": "[xyz] (optional paramater, default is xxx",  "b1iadminPassword": "[password for B1iadmin user (optional parameter, default is empty password)]",  "fqdnMapping": "[0|1 (optional parameter, defaults is 0)]",  "smbhubDomainname": "[external DNS Name (optional parameter, default is global default)]"  } |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### update

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Updates a client to the latest available image. The optional parameters appImageName, dbImageName and imageTag can be supplied with a valid combination of values retuned by ls\_app\_images and ls\_db\_images. Default values will be used if the optional parameters are omitted, These Default values are the latest entries in the image list files (under /opt/sap/B1iFCloudManager/etc) in alphabetical order. |
| Request | {  " tenantId": "[Tenant ID]",  appImageName: "[Name of the Application Image to be used (optional parameter)]",  dbImageName: "[Name of the Database Image to be used (optional parameter)]",  imageTag: "[Tag of of both Images (optional parameter)]"  } |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### start

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Starts a stopped client |
| Request | {  " tenantId": "[Tenant ID]",  } |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### stop

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Stops a running client |
| Request | {  " tenantId": "[Tenant ID]",  } |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### rm

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Removes a client identified by tenantId. The method deletes the client without further confirmation. After successfully calling the method, call config\_proxy. |
| Request | {  " tenantId": "[Tenant ID]",  } |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### save

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Creates a client backup identified by the tenantId field of the request payload |
| Request | {  " tenantId": "[Tenant ID]",  } |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### ls\_backups

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Lists all backups that have been created by calling endpoint save or command save in tenantShell |
| Request | Not required |
| Response | {  "backups": [  {  "name": [Name of backup]  }  ],  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### restore

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Restores backups that have been created either by endpoint save or vy tenantShell command save |
| Request | {  " backupName": "[Name of backup as returned by ls\_backups]",  } |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### create\_proxy

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Creates the internal reverse proxy |
| Request | Not required |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### config\_proxy

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Configures the internal reverse proxy by adding or removing forwarding rules for added or removed clients.  Call the method after a successful create or rm call. |
| Request | Not required |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### stop\_proxy

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Stops the internal reverse proxy. Do not call the endpoint, if the REST server runs behind the reverse proxy. |
| Request | Not required |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### start\_proxy

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Starts the internal reverse proxy |
| Request | Not required |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### rm\_proxy

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Removes the internal reverse proxy and configuration data without further warning. Do not call the endpoint, if the REST server runs behind the reverse proxy |
| Request | Not required |
| Response | {  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### decorate

|  |  |
| --- | --- |
| Method | Description |
| **POST** | Decorates a string to be used as password string in a create request |
| Request | {  " string": "[String to be decorated]",  } |
| Response | {  " string": "[the decorated staring]",  "stdout": [  "[stdout of background process]",  …  ],  "exitcode": [exit code of background process],  "stderr": [  [stderr of background process],  …  ],  "info": "[okay]|[any other info text]"  } |

#### ls\_app\_images

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Lists available Application images |
| Request | Not required |
| Response | {  "stdout": [  "stdout of background process"  ],  "images": [  {  "name": "image name",  "tag": "Tag of image",  }  ],  "exitcode": exit code of background process,  "stderr": [  "stderr of background process"  ],  "info": "[okay]|[any other info text]"  } |

#### ls\_db\_images

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Lists available Database images |
| Request | Not required |
| Response | {  "stdout": [  "stdout of background process"  ],  "images": [  {  "name": "image name",  "tag": "Tag of image",  }  ],  "exitcode": exit code of background process,  "stderr": [  "stderr of background process"  ],  "info": "[okay]|[any other info text]"  } |

#### ls\_proxy\_images

|  |  |
| --- | --- |
| Method | Description |
| **GET** | Lists available Reverse Proxy images |
| Request | Not required |
| Response | {  "stdout": [  "stdout of background process"  ],  "images": [  {  "name": "image name",  "tag": "Tag of image",  }  ],  "exitcode": exit code of background process,  "stderr": [  "stderr of background process"  ],  "info": "[okay]|[any other info text]"  } |

|  |
| --- |
|  |
| www.sap.com/contactsap |
| .  © 2020 SAP SE or an SAP affiliate company. All rights reserved. No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary. These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty. SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. Please see http://www.sap.com/corporate-en/legal/copyright/index.epx for additional trademark information and notices. |

SAPLogo.gif