**Software Design Description (SDD)**

**Template Version: 2.04**

Changes to 2.0.4

All links to portal have been updated

Header and Footer updated

New approach to document versioning introduced

Chapter 5 completely revised (FSBP design template provided the basis)

Chapter 9 completely revised (ABAP Unit Design Template provided the basis)

Chapter 5.5 introduced Traceability via reference table to SRS

Chapter 4 references to Technical Architecture Modeling (TAM) standard implemented

2.03: links to Portal

2.02: Enterprise Service Oriented Architecture (Enterprise SOA) added to Product Standards; footer corrected

2.01: ARB decision: chapter 5.3 User Interfaces removed; merge of subchapters 5.1, 5.2, 5.3 to 5.  
 English language check

**How to Use this Template**

Do not delete the non-applicable and optional sections, but mark them with “Not relevant” and give a short explanation as to why they are not relevant. Additional chapters can be added, if required.

**Document Versioning**

a) Documents versions follow the scheme x.yz.

b) When you change a document, create a new version and keep the old document version.

c) There are three types of changes, which affect the version number as follows

1. Status Changes



2. Major Changes

Increment y-part of document version for complete revisions (e.g. 1.3 to 1.4)

3. Minor Changes

Increment z-part of document version for minor corrections (e.g. 1.3 to 1.3.1)

This document contains a macro that maintains the general fields automatically and is started by using the button .

To review documents based on this template use the [Review Checklist for Software Design Descriptions](https://portal.wdf.sap.corp/irj/servlet/prt/portal/prtroot/com.sap.km.cm.docs/guid/6278bee7-0501-0010-da85-d9841a07115b)

Aim to store your documents in document management systems such as cProjects/cFolders and avoid entering links to documents on shares or servers.

If appropriate, insert graphics to explain complex models or dependencies. Specify the exact source of the graphics so that the knowledge management team can use it.

The design document has to be available in the English language. Other languages are optional.

Before the external review, this document should be reviewed by a native speaker.

See also the document link [Relation of PIL Documents](http://service.sap.com/~sapidb/011000358700001933802005) for further information.

This template (Software Design Description) replaces the former template for design.

It should be used for the software design of:

SAP components according to the solution builder (transaction ZSOLPROD in ISP) or the Product Planning Management System (PPMS, transaction BOPP in ISP)

Subcomponents such as application components in the application component hierarchy (ACH), Java packages, or other types of programs or subprograms

Original Author: [Christian Schmidkonz](http://tel.wdf.sap.corp:1080/cgi-bin/tel.cgi?D020317)

Template Owner: Markus Leins; last changed 12/05/2006

Contact the template owner if you have any ideas to improve this template.

|  |  |  |  |
| --- | --- | --- | --- |
| Version: 0.1 | **Before you start to work with this template read the instructions (see link) “How to Use this Template” on the first page** | | |
| Author: Haifeng Yao |  | | |
|  | EC2 Manager Design | | |
|  | | |
|  | **History** | | |
|  | Version  Enter document version; (see versioning rules above) | Date  Enter document version date 2005-12-21 | Status (Comments)  Draft, For Review, Released.  Comments are needed if a SDD has several “releases” |
|  | 0.1 | 2022-07 | Created |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Glossary 2](#_Toc107658026)

[2 Detailed design 3](#_Toc107658027)

[2.1 Key requirements and design goals 3](#_Toc107658028)

[2.2 Technical interpretation 3](#_Toc107658029)

[2.2.1 Stateless architecture 3](#_Toc107658030)

[2.2.1.1 Why serverless architecture 3](#_Toc107658031)

[2.2.1.2 Advantage of serverless architecture 3](#_Toc107658032)

[2.2.1.3 Disadvantage of serverless architecture 4](#_Toc107658033)

[2.2.2 Technology chosen 5](#_Toc107658034)

[2.2.3 Security control 6](#_Toc107658035)

[2.2.4 Limitations 6](#_Toc107658036)

[2.3 Configuration 7](#_Toc107658037)

[2.3.1 Create CloudFormation stack 7](#_Toc107658038)

[2.3.2 AWS Systems Manager 7](#_Toc107658039)

[2.3.3 API Gateway 7](#_Toc107658040)

Note: Specific hints on how to proceed with cARC/cProjects are described in (see link) **How to work with this document** on the first page of this template.

**R**: Required Entry; For Roles and Responsibilities see also the Q&A in the Project Weaver: Compare quick link: [**/projectweaver**](https://portal.wdf.sap.corp/irj/servlet/prt/portal/prtroot/com.sap.km.cm.docs/Infocenters/WS%20PTG/PTG/Standards%20%26%20Tools/PIL%20Tools/ProjectWeaver/index.html) -> see Navigation pad “Media Library” and open ProjectWeaver

# Glossary

Use generally known expressions and those technical terms that are defined in the terminology database of SAP.

Enter all terms with significant importance for this document/development.

Working together with your information development team, gather, standardize, and define all new or changed terms and abbreviations in English (and German) required to properly understand the SRS. Provide draft definitions of these terms. Refer to entries in the appendix of the SRS, to entries in the SAP term terminology database (see quick link [**/go/sapterm**](https://portal.wdf.sap.corp/go/sapterm)), or to other documents.

Make sure that all abbreviations, initials and acronyms are clearly associated with their relevant terms. Define no more than one allowed abbreviation per term. For more information, see section “Abbreviations on the User Interface” in the general [*Standards and Guidelines for Writing at SAP*](http://intranet.sap.com/~sapidb/011000358700001398042004E) (or see link Product Standard [**Documentation**](https://sapportal.wdf.sap.corp/irj/portal?NavigationTarget=ROLES://portal_content/com.sap.sen.ptg.workspace.com_sap_sen_ptg_workspace/com.sap.sen.ptg.workspace.f_roles/com.sap.sen.ptg.workspace.rl_ptg/f_ptg/com.sap.sen.ptg.workspace.pg_standards_and_tools&)).

|  |  |
| --- | --- |
| **Term** | **Definition** |
| SaaS | Software as a service (SaaS) is a way of delivering applications remotely over the internet instead of locally on machines (known as “on-premise” software). |
| FaaS | Function as a service |
| AWS | Amazon Web Services (AWS) is a [Amazon](https://en.wikipedia.org/wiki/Amazon.com) cloud platform that provides [on-demand](https://en.wikipedia.org/wiki/Software_as_a_service) [cloud computing](https://en.wikipedia.org/wiki/Cloud_computing) [platforms](https://en.wikipedia.org/wiki/Computing_platform) and [APIs](https://en.wikipedia.org/wiki/Application_programming_interface). |
| [EC2](https://aws.amazon.com/ec2/) | EC2 (Elastic Compute Cloud) is a web service that provides resizable computing capacity—literally, servers in Amazon's data centers—that you use to build and host your software systems. |
| [Lambda](https://aws.amazon.com/lambda/) | AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. |
| [Step Functions](https://aws.amazon.com/step-functions) | Step Functions is a serverless orchestration service that lets you easily coordinate multiple Lambda functions into flexible workflows that are easy to debug and easy to change. |
| [API Gateway](https://aws.amazon.com/api-gateway/?e=gs2020&p=deepdiveserverless) | Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. |
| [IAM](https://aws.amazon.com/iam/) | AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources. |
| SAM | Serverless Application Model (SAM) is a developer tools which helps simplify deployment and testing of your Lambda functions and serverless applications. |
| [CloudFormation](https://docs.aws.amazon.com/cloudformation/index.html) | AWS CloudFormation is a service that helps you modeland set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. |
| Stack | CloudFormation Stack is a collection of AWS resources you want to deploy together as a group. |
| [ARN](https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html) | Amazon Resource Names (ARNs) uniquely identify AWS resources. We require an ARN when you need to specify a resource unambiguously across all of AWS |
| [Systems Manager](https://aws.amazon.com/systems-manager/) | AWS Systems Manager is a management solution that centralizes operational data from multiple AWS services and automates tasks across AWS resources. |
| SSM Agent | AWS Systems Manager Agent (SSM Agent) is Amazon software that runs on Amazon EC2 instances, edge devices, and on-premises servers and virtual machines. SSM Agent makes it possible for Systems Manager to update, manage, and configure the resources. |

# Detailed design

## Key requirements and design goals

Our team manages a fleet of EC2 instances based on Amazon Linux 2. The number of instances has grown significantly over the past few months. As a consequence, the workload of managing these instances manually has become a huge burden for the team.

To reduce the workload, we would like to create a new **AWS application** which is called **EC2 Manager** to automate frequently recurring tasks. The application should have below functions for the first version.

1. Retrieve information about resources in certain AWS account (e.g., get a list of running EC2 instances).
2. Run short tasks, no longer than 10 minutes, on the operating system layer of an EC2 instance (e.g., a Linux shell script).
3. Run longer tasks of up to 2 hours on the operating system layer of an EC2 instance (e.g., an application update).

## Technical interpretation

### Stateless architecture

To build better application and release it faster, we decided to use stateless architecture (aka. *function as a service*, FaaS) which is a software design pattern where applications are hosted by a third-party service, eliminating the need for server software and hardware management by the developer. Applications are broken up into individual functions that can be invoked and scaled individually.

#### Why serverless architecture

Hosting a software application on the internet usually involves managing some kind of server infrastructure. Typically, this means a virtual or physical server that needs to be managed, as well as the operating system and other web server hosting processes required for the application to run. Using a virtual server from a cloud provider such as [Amazon](https://aws.amazon.com/) does eliminate the physical hardware concerns, but still requires some level of management of the operating system and the web server software processes.

With a serverless architecture, the development team focus purely on the individual functions in the application code. Services such as AWS Step Functions or [AWS Lambda](https://aws.amazon.com/lambda/) take care of all the physical hardware, virtual machine operating system, and web server software management. You only need to worry about your code.

#### Advantage of serverless architecture

Comparing to architecture that relies on backend servers, serverless architecture has below advantages:

* Faster release

Eliminate operational overhead so the development team can release quickly, get feedback, and iterate to get to market faster.

* Lower cost

Lower cost when factoring in the reduced need for DevOps staff to set-up and maintain infrastructure over time and the pay per usage billing meaning low traffic periods can cost nothing.

* Higher scalability

With technologies that automatically scale from zero to peak demands, adaption to customer needs is faster than ever.

* Easier development

Serverless applications have built-in service integrations, so the development team can focus on building the application instead of configuring it. Development velocity increases drastically since it is a lot easier to put solutions together and deploy it into production

* Higher availability

Increased up time since the cloud manages all the services for you as well as manages better under unexpected load.

#### Disadvantage of serverless architecture

At the same time, serverless architecture has some disadvantages.

* Operation difficulty

Increased difficulty in the areas of debugging and monitoring. The developers are dependent on vendors for debugging and monitoring tools. debugging distributed systems is difficult and usually requires access to a significant number of relevant metrics to identify the root cause.

* Possible vendor lock-in

It’s very likely that whatever Serverless features you’re using from one vendor will be implemented differently by another vendor. If you want to switch vendors, you’ll almost certainly need to update your operational tools (deployment, monitoring, etc.), you’ll probably need to change your code (e.g., to satisfy a different FaaS interface), and you may even need to change your design or architecture if there are differences to how competing vendor implementations behave.

* Architectural complexity

Decisions about how small (granular) the function should be, takes time to assess, implement and test. There should be a balance between the number of functions should an application call. It gets cumbersome to manage too many functions and ignoring granularity will end up creating mini-monoliths. One should especially consider using a serverless provider if the number of functions need to be hosted is small.  
  
Considering this application is relatively simple and the limited team resource, we favor to use stateless architecture.

### Technology chosen


                Diagram showing how Systems Manager capabilities use a similar process of set up,
                    launching, processing, and reporting.
            

Serverless applications are generally built using fully managed services as building blocks across the compute, data, messaging and integration, streaming, and user management and identity layers. Based on serverless architecture guidelines, below AWS services are chosen.

* AWS Systems Manager for tasks automation

Amazon Web Services Systems Manager is a collection of capabilities that helps you automate management tasks such as collecting system inventory, applying operating system (OS) patches, automating the creation of Amazon Machine Images (AMIs), and configuring operating systems (OSs) and applications at scale. Systems Manager lets you remotely and securely manage the configuration of your managed nodes. A managed node is any Amazon Elastic Compute Cloud (Amazon EC2) instance, edge device, or on-premises server or virtual machine (VM) that has been configured for Systems Manager.

* AWS Lambda functions for task execution

AWS Lambda is a serverless, event-driven compute service that lets you run stateless serverless applications on a managed platform that supports microservices architectures, deployment, and management of execution at the function layer.

* AWS Step Functions for workflow orchestration

Step Functions is a serverless orchestration service that lets you easily coordinate multiple Lambda functions into flexible workflows that are easy to debug and easy to change.

[Standard workflow](https://docs.aws.amazon.com/step-functions/latest/dg/concepts-standard-vs-express.html)  is used since the workflow needs to wait for a human approval which might take more than five minutes to respond.

* Amazon API Gateway as application endpoint

Amazon API Gateway is used as application endpoint and to associate the AWS Step Functions APIs. When an HTTPS request is sent to an API method, API Gateway invokes the Step Functions API action.

* Java as programming language

The [AWS SDK for Java](https://docs.aws.amazon.com/sdk-for-java/) provides a Java API for Amazon Web Services infrastructure services. Using the SDK, you can build applications on top of Amazon S3, Amazon EC2, Amazon DynamoDB, and more. [AWS Systems Manager Java code examples](https://github.com/awsdocs/aws-doc-sdk-examples/tree/main/javav2/example_code/ssm)  is referred for the implementation.

### Security control

To secure access to AWS services, IAM roles need to be created and assigned. For example Lambda functions use an execution role to get permission to write logs to Amazon CloudWatch Logs, and to access other services and resources.

Below roles are created for this application:

|  |  |  |
| --- | --- | --- |
| **Role** | **Policies** | **ARN (Amazon Resource Name)**[Development Architect]  [Consultant]  [IMS Developer] [User Interface Designer]  [Info Developer] [Knowledge Management Contact] [Solution Management Expert] [topic specialist of area] [Translator]  [Modeler]  [Producer of training material] |
| LambdaBasicExecutionRole | AWSLambdaEdgeExecutionRole | arn:aws:iam::270078487780:role/service-role/LambdaBasicExecutionRole |
| APIGatewayToStepFunctionsRole | AmazonAPIGatewayPushToCloudWatchLogs  AWSStepFunctionsFullAccess | arn:aws:iam::270078487780:role/APIGatewayToStepFunctions |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

CheckEC2InstanceInfo

### Limitations

Explain limitations of the current solution and propose how you may be able to improve the  
solution.

* Fine-grained security control for operation of different instance
* Application log
* Audit log
* Auto-scaling
* Missing UI
* Source control, automate deployments are missing
* Immediate task execution feedback (wait for callback) is not provided yet
* Dynamic Lambda function determination based on parameter not supported
* Retry in case of error is missing

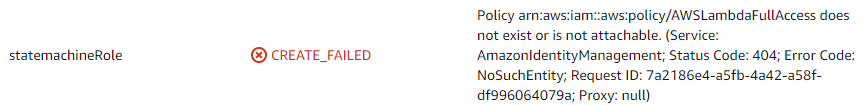
## Configuration

### Create CloudFormation stack

[Create a CloudFormation stack](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-console-create-stack.html) in the us-east-1 (N. Virginia) region with the provided template. CloudFormation will create several resources in the target AWS account, it will be used as a foundation to complete the project. This includes an EC2 instance, a Step Function and Lambda functions.

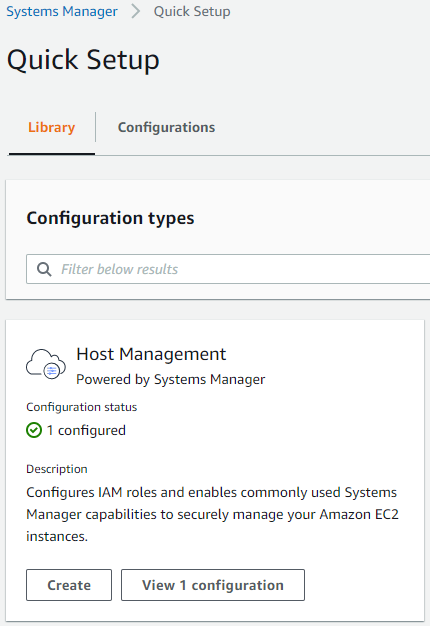
**Note:**

* AWSLambdaFullAccess does not exist, it’s replaced by service-role/AWSLambdaBasicExecutionRole instead in the template to create the stack.



### AWS Systems Manager

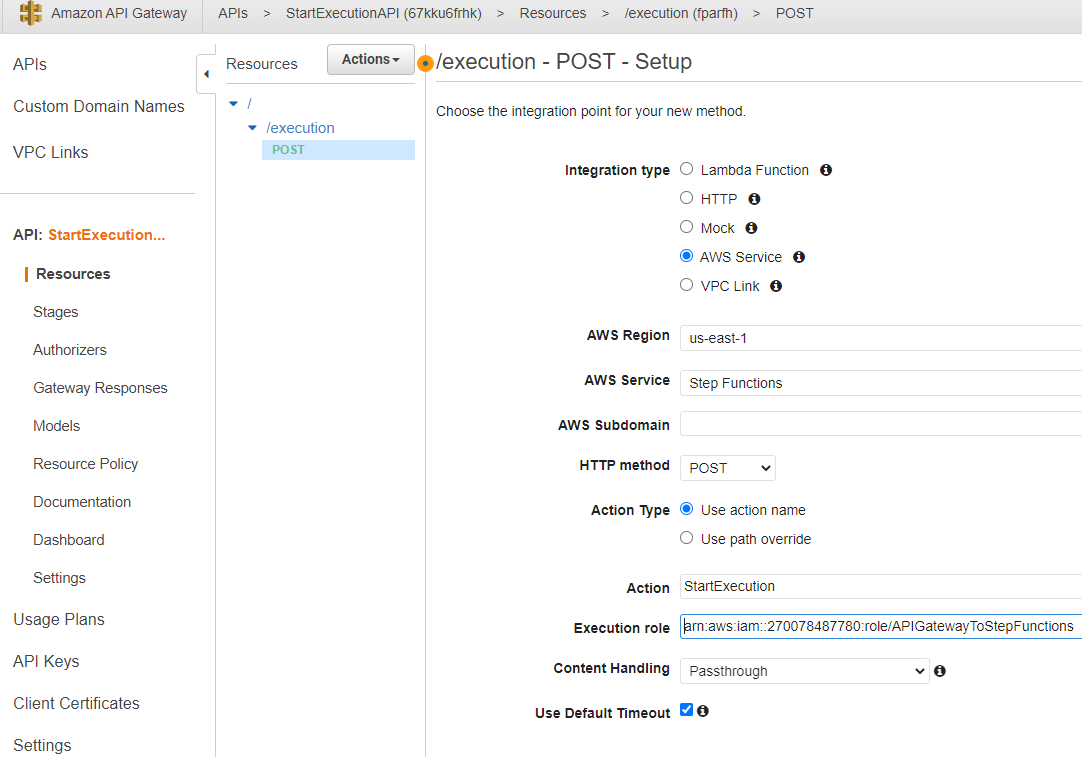
As a prerequisites, [Setting up AWS Systems Manager](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-setting-up.html) need to be performed first. After these steps are complete, users in your organization can use Systems Manager to configure, manage, and access the *managed nodes*.  [AWS Systems Manager Quick Setup](https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-quick-setup.html) is used for a quicker setup. During the quick setup, AWS Systems Manager Agent (SSM Agent) is installed in the managed node, this makes it possible for Systems Manager to update, manage, and configure these resources.



### API Gateway

Amazon API Gateway is used to associate the AWS Step Functions APIs with methods in an API Gateway API. When an HTTP request is sent to an API method, API Gateway invokes your Step Functions API action.

[Creating a Step Functions API Using API Gateway](https://docs.aws.amazon.com/step-functions/latest/dg/tutorial-api-gateway.html)



Describe anything that is relevant to the design, but could not integrated in the chapters above.

List your relevant solution-specific design guidelines, programming guidelines, checklists.