

Developer Operations

AWS Well-Architected Framework

What is the AWS Well-Architected Framework?

- The goal of this framework is to help cloud architects to build the most secure, high-performing, resilient, and efficient infrastructure possible for their applications
- It provides a set of questions developed by AWS experts to help customers think critically about their architecture.
- It asks, "Does your infrastructure follow best practices?"

AWS Well-Architected Outcomes

Architects should leverage the AWS Well-Architected Framework in order to:

- Increase awareness of architectural best practices.
- Address foundational areas that are often neglected.
- Evaluate architectures using a consistent set of principles.

AWS Well-Architected Guidance

- The AWS Well-Architected Framework does not provide:
 - Implementation details
 - Architectural patterns
 - Relevant case studies
- However, it does provide:
 - Questions centered on critically understanding architectural decisions
 - Services and solutions relevant to each question
 - References to relevant resources
 - Well Architected Tool

<https://docs.aws.amazon.com/wellarchitected/latest/userguide/intro.html>

AWS Well-Architected Framework Documentation

- White papers:
 - AWS Well-Architected Framework (June 2018)
 - Security Pillar (July 2018)
 - Reliability Pillar (September 2018)
 - Performance Efficiency Pillar (July 2018)
 - Cost Optimization Pillar (July 2018)
 - Operational Excellence Pillar (July 2018)

Pillars Of The Well-Architected Framework

Security

Protect information and systems.



Reliability

Recover from failure and mitigate disruption.



Performance Efficiency

Use resources sparingly.



Cost Optimization

Eliminate unneeded expense.



Operational Excellence

Manage and monitor.



Security

Security

Protect
information
and systems



- The ability to protect:
 - Information
 - Systems
 - Assets
- While delivering business value through:
 - Risk assessments
 - Mitigation strategies

Security Pillar Principles

- 📦 Apply Security at All Layers
- 📦 Enable Traceability
- 📦 Automate Responses To Security Events
- 📦 Focus On Securing Your System
- 📦 Automate Security Best Practices



Best Practice: Secure Your Infrastructure Everywhere

Build security into every layer of your infrastructure.

Physical data centers typically rely on security at the perimeter. AWS enables you to implement security at the perimeter as well as **within and between your resources**.

Things to consider:

- ❏ Isolate parts of your infrastructure
- ❏ Encrypt data in transit and at rest
- ❏ Enforce access control granularly, using the principle of least privilege
- ❏ Use multi-factor authentication
- ❏ Leverage managed services
- ❏ Log access of resources
- ❏ Automate your deployments to keep security consistent

Well-Architected Pillar 1: Security

- The ability to protect information, systems, and assets while delivering business value through risk assessments and mitigation strategies.
 - Identity and access management
 - Detective controls
 - Infrastructure protection
 - Data protection
 - Incident response



Reliability

Reliability

Recover from failure and mitigate disruption.



The ability of a system to:

- ❏ Recover from infrastructure or service failures
- ❏ Dynamically acquire computing resources to meet demand
- ❏ Mitigate disruptions such as:
 - Misconfigurations
 - Transient network issues






Reliability: Design Principles



- Test recovery procedures
- Automatically recover from failure
- Scale horizontally to increase aggregate system availability
- Stop guessing capacity
- Manage change in automation

Key Services For Reliability



Areas	Key Services	
Foundations	 AWS IAM	 Amazon VPC
Change management	 AWS CloudTrail	 AWS Config
Failure management	 AWS CloudFormation	

Performance Efficiency

Performance Efficiency

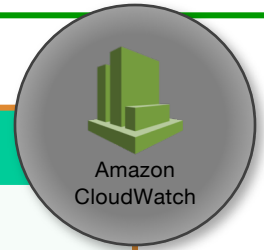
Use resources sparingly.










The ability to:

- ❏ Use computing resources efficiently to meet system requirements
- ❏ Maintain that efficiency as demand changes and technologies evolve

Key Services For Performance Efficiency



Areas	Key Services		
Compute	 Auto Scaling		
Storage	 Amazon EBS	 Amazon S3	 Amazon Glacier
Database	 Amazon RDS	 Amazon DynamoDB	
Space-time trade-off	 Amazon CloudFront		

Cost Optimization

Cost Optimization

Eliminate
unnecessary
expense.



The ability to avoid or eliminate:

- 📦 Unneeded cost
- 📦 Suboptimal resources

Best Practice: Optimize For Cost

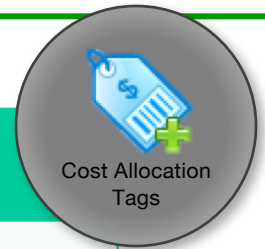
Take advantage of AWS's flexible platform to increase your cost efficiency.








Ensure that your resources are **sized appropriately**, that they **scale in and out** based on need, and that you're taking advantage of **different pricing options**.

Things to consider:

- 📦 Are my resources the right size and type for the job?
- 📦 What metrics should I monitor?
- 📦 How do I make sure that resources not in use are turned off?
- 📦 How often will I need to use this resource?
- 📦 Can I replace any of my servers with managed services?

Key Services For Cost Optimization



Areas	Key Services	
Matched supply and demand	 Auto Scaling	
Cost-effective resources	 Spot and Reserved Instances	 AWS Cost Explorer
Expenditure awareness	 Amazon CloudWatch	 Amazon SNS
Optimizing over time	 AWS Blog & What's New	 AWS Trusted Advisor

Operational Excellence

Operational Excellence

Manage and monitor.



The ability to:

- 📦 Successfully manage daily operations
- 📦 Manage and automate changes
- 📦 Respond to events

EXTRA : CloudFormation

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>

Anatomy Of A CloudFormation Template

```
"Description" : "JSON  
string",
```

```
"Metadata" : {  
  template metadata  
},  
"Parameters" : {  
  set of parameters  
},  
"Mappings" : {  
  set of mappings },  
"Conditions" : {  
  set of conditions  
},  
"Resources" : {  
  set of resources },  
"Outputs" : {  
  set of outputs }
```

Description:

Text string that describes the template

- Literal string between 0 and 1024 bytes long
- Cannot use a parameter or function to specify it

```
"Description" : "This template builds a  
VPC with one public and one private  
subnet",
```

JSON example

Anatomy Of A CloudFormation Template

```
"Description" : "JSON
string",
"Metadata" : {
  template metadata
},
"Parameters" : {
  set of parameters
},
"Mappings" : {
  set of mappings },
"Conditions" : {
  set of conditions
},
"Resources" : {
  set of resources },
"Outputs" : {
  set of outputs }
```

JSON example

Resources:

Services (and their settings) that you want to launch in the stack

Properties:

- Each resource must be declared separately (except multiple instances of the same resource)
- Resource declaration has the resource's attributes

```
"Resources" : {
  "Logical ID" : {
    "Type" : "Resource type",
    "Properties" : {
      Set of properties } } }
```

Anatomy Of CloudFormation Template: Resources

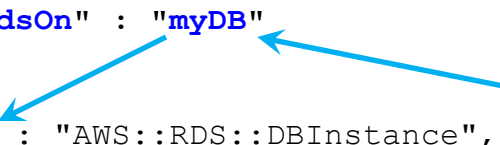
```
"Resources" : {  
  "MyInstance" : {  
    "Type" : "AWS::EC2::Instance",  
    "Properties" : {  
      "UserData" : {  
        "Fn::Base64" : {  
          "Fn::Join" : [ "", [ "Queue=", { "Ref" : "MyQueue" } ] ]  
        } },  
      "AvailabilityZone" : "us-east-1a",  
      "ImageId" : "ami-20b65349" }  
    },  
  "MyQueue" : {  
    "Type" : "AWS::SQS::Queue",  
    "Properties" : { } } } }
```

JSON example

Resource Attribute: DependsOn

The "DependsOn" attribute specifies that the creation of a specific resource follows another. You can use the DependsOn attribute with any resource.

```
"Resources" : {  
  "AppServerInstance" : {  
    "Type" : "AWS::EC2::Instance",  
    "Properties" : {  
      "ImageId" : {  
        "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]  
      }  
    },  
    "DependsOn" : "myDB"  
  },  
  "myDB" : {  
    "Type" : "AWS::RDS::DBInstance",  
    "Properties" : {  
      ...  
    }  
  }  
}
```

A blue arrow points from the "myDB" resource definition to the "DependsOn" attribute of the "AppServerInstance" resource, indicating that the EC2 instance depends on the RDS database instance.

Creates the Amazon EC2 instance **only after** the RDS database instance has been created.

JSON example

When A DependsOn Attribute Is Required

The following resources depend on a VPC gateway attachment when they have an associated public IP address and are in a VPC:

- Auto Scaling group
- Amazon EC2 instances
- Elastic Load Balancing load balancers
- Elastic IP address
- Amazon RDS database instances
- Amazon VPC routes that include the Internet gateway

Special Resource: Wait Condition

Wait conditions are special CloudFormation resources that pause the creation of the stack and wait for a signal before it continues.

Use a wait condition to coordinate the creation of stack resources with other configuration actions external to the stack creation.

```
"myWaitCondition" : {  
  "Type" : "AWS::CloudFormation::WaitCondition",  
  "DependsOn" : "Ec2Instance",  
  "Properties" : {  
    "Handle" : { "Ref" : "myWaitHandle" },  
    "Timeout" : "4500"  
  }  
}
```

JSON example

Wait condition that begins after the successful creation of the “Ec2Instance” resource

Anatomy Of A CloudFormation Template

```
"Description" : "JSON  
string",
```

```
"Metadata" : {  
    template metadata  
},
```

```
"Parameters" : {  
    set of parameters  
},
```

```
"Mappings" : {  
    set of mappings },
```

```
"Conditions" : {  
    set of conditions  
},
```

```
"Resources" : {  
    set of resources },
```

```
"Outputs" : {  
    set of outputs }
```

Parameters:

Values you can pass in to your template at runtime

- Allow stacks to be customized at launch of a template
- Can specify allowed and default values for each parameter

CloudFormation Template: Parameters Example

```
"Parameters" : {  
  "InstanceTypeParameter" : {  
    "Type" : "String",  
    "Default" : "t2.micro",  
    "AllowedValues" : ["t2.micro", "m1.small", "m1.large"],  
    "Description" : "Enter t2.micro, m1.small, or m1.large. Default  
      is t2.micro." } }  
}
```

```
"Resources" : {  
  "WebAppInstance" : {  
    "Type" : "AWS::EC2::Instance",  
    "Properties" : {  
      "InstanceType" : { "Ref" : "InstanceTypeParameter" },  
      "ImageId" : "ami-2f726546"  
    }  
  }  
}
```

CloudFormer

CloudFormer is a template creation beta tool that creates an AWS CloudFormation template from existing AWS resources in your account. You select any supported AWS resources that are running in your account, and CloudFormer creates a template in an Amazon S3 bucket.

Use CloudFormer to produce templates that you can use as a starting point. Not all AWS resources or resource properties are supported.

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/cfn-using-cloudformer.html>