

operations in case a later operation of the same transaction fails. Here, the core function is maintaining consistency.

- Time critical systems should be able to deal with dependencies not responding in a timely manner. In these cases, the circuit breaker pattern can be used. When responses from a dependency start timing out, the system can switch to a closed state where no additional call are made.
- An application may read parameters from a parameter store. It can be useful to create container images with a default set of parameters and use these in case the parameter store is unavailable.

Note that the pathways taken in case of component failure need to be tested and should be significantly simpler than the primary pathway. Generally, [fallback strategies should be avoided](#).

Implementation steps

Identify external and internal dependencies. Consider what kinds of failures can occur in them. Think about ways that minimize negative impact on upstream and downstream systems and customers during those failures.

The following is a list of dependencies and how to degrade gracefully when they fail:

1. **Partial failure of dependencies:** A component may make multiple requests to downstream systems, either as multiple requests to one system or one request to multiple systems each. Depending on the business context, different ways of handling for this may be appropriate (for more detail, see previous examples in Implementation guidance).
2. **A downstream system is unable to process requests due to high load:** If requests to a downstream system are consistently failing, it does not make sense to continue retrying. This may create additional load on an already overloaded system and make recovery more difficult. The circuit breaker pattern can be utilized here, which monitors failing calls to a downstream system. If a high number of calls are failing, it will stop sending more requests to the downstream system and only occasionally let calls through to test whether the downstream system is available again.
3. **A parameter store is unavailable:** To transform a parameter store, soft dependency caching or sane defaults included in container or machine images may be used. Note that these defaults need to be kept up-to-date and included in test suites.
4. **A monitoring service or other non-functional dependency is unavailable:** If a component is intermittently unable to send logs, metrics, or traces to a central monitoring service, it is often

best to still execute business functions as usual. Silently not logging or pushing metrics for a long time is often not acceptable. Also, some use cases may require complete auditing entries to fulfill compliance requirements.

5. A primary instance of a relational database may be unavailable: Amazon Relational Database Service, like almost all relational databases, can only have one primary writer instance. This creates a single point of failure for write workloads and makes scaling more difficult. This can partially be mitigated by using a Multi-AZ configuration for high availability or Amazon Aurora Serverless for better scaling. For very high availability requirements, it can make sense to not rely on the primary writer at all. For queries that only read, read replicas can be used, which provide redundancy and the ability to scale out, not just up. Writes can be buffered, for example in an Amazon Simple Queue Service queue, so that write requests from customers can still be accepted even if the primary is temporarily unavailable.

Resources

Related documents:

- [Amazon API Gateway: Throttle API Requests for Better Throughput](#)
- [CircuitBreaker \(summarizes Circuit Breaker from “Release It!” book\)](#)
- [Error Retries and Exponential Backoff in AWS](#)
- [Michael Nygard “Release It! Design and Deploy Production-Ready Software”](#)
- [The Amazon Builders' Library: Avoiding fallback in distributed systems](#)
- [The Amazon Builders' Library: Avoiding insurmountable queue backlogs](#)
- [The Amazon Builders' Library: Caching challenges and strategies](#)
- [The Amazon Builders' Library: Timeouts, retries, and backoff with jitter](#)

Related videos:

- [Retry, backoff, and jitter: AWS re:Invent 2019: Introducing The Amazon Builders' Library \(DOP328\)](#)

Related examples:

- [Well-Architected lab: Level 300: Implementing Health Checks and Managing Dependencies to Improve Reliability](#)

REL05-BP02 Throttle requests

Throttle requests to mitigate resource exhaustion due to unexpected increases in demand. Requests below throttling rates are processed while those over the defined limit are rejected with a return a message indicating the request was throttled.

Desired outcome: Large volume spikes either from sudden customer traffic increases, flooding attacks, or retry storms are mitigated by request throttling, allowing workloads to continue normal processing of supported request volume.

Common anti-patterns:

- API endpoint throttles are not implemented or are left at default values without considering expected volumes.
- API endpoints are not load tested or throttling limits are not tested.
- Throttling request rates without considering request size or complexity.
- Testing maximum request rates or maximum request size, but not testing both together.
- Resources are not provisioned to the same limits established in testing.
- Usage plans have not been configured or considered for application to application (A2A) API consumers.
- Queue consumers that horizontally scale do not have maximum concurrency settings configured.
- Rate limiting on a per IP address basis has not been implemented.

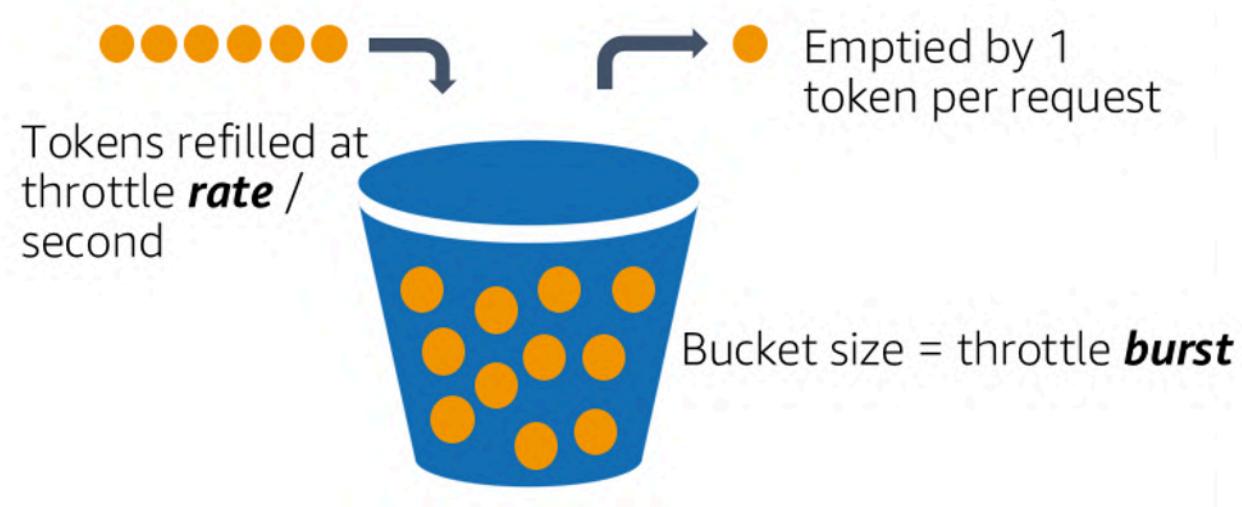
Benefits of establishing this best practice: Workloads that set throttle limits are able to operate normally and process accepted request load successfully under unexpected volume spikes. Sudden or sustained spikes of requests to APIs and queues are throttled and do not exhaust request processing resources. Rate limits throttle individual requestors so that high volumes of traffic from a single IP address or API consumer will not exhaust resources impact other consumers.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Services should be designed to process a known capacity of requests; this capacity can be established through load testing. If request arrival rates exceed limits, the appropriate response signals that a request has been throttled. This allows the consumer to handle the error and retry later.

When your service requires a throttling implementation, consider implementing the token bucket algorithm, where a token counts for a request. Tokens are refilled at a throttle rate per second and emptied asynchronously by one token per request.



The token bucket algorithm.

[Amazon API Gateway](#) implements the token bucket algorithm according to account and region limits and can be configured per-client with usage plans. Additionally, [Amazon Simple Queue Service \(Amazon SQS\)](#) and [Amazon Kinesis](#) can buffer requests to smooth out the request rate, and allow higher throttling rates for requests that can be addressed. Finally, you can implement rate limiting with [AWS WAF](#) to throttle specific API consumers that generate unusually high load.

Implementation steps

You can configure API Gateway with throttling limits for your APIs and return 429 Too Many Requests errors when limits are exceeded. You can use AWS WAF with your AWS AppSync and API Gateway endpoints to enable rate limiting on a per IP address basis. Additionally, where your system can tolerate asynchronous processing, you can put messages into a queue or stream to speed up responses to service clients, which allows you to burst to higher throttle rates.

With asynchronous processing, when you've configured Amazon SQS as an event source for AWS Lambda, you can [configure maximum concurrency](#) to avoid high event rates from consuming available account concurrent execution quota needed for other services in your workload or account.

While API Gateway provides a managed implementation of the token bucket, in cases where you cannot use API Gateway, you can take advantage of language specific open-source implementations (see related examples in Resources) of the token bucket for your services.

- Understand and configure [API Gateway throttling limits](#) at the account level per region, API per stage, and API key per usage plan levels.
- Apply [AWS WAF rate limiting rules](#) to API Gateway and AWS AppSync endpoints to protect against floods and block malicious IPs. Rate limiting rules can also be configured on AWS AppSync API keys for A2A consumers.
- Consider whether you require more throttling control than rate limiting for AWS AppSync APIs, and if so, configure an API Gateway in front of your AWS AppSync endpoint.
- When Amazon SQS queues are set up as triggers for Lambda queue consumers, set [maximum concurrency](#) to a value that processes enough to meet your service level objectives but does not consume concurrency limits impacting other Lambda functions. Consider setting reserved concurrency on other Lambda functions in the same account and region when you consume queues with Lambda.
- Use API Gateway with native service integrations to Amazon SQS or Kinesis to buffer requests.
- If you cannot use API Gateway, look at language specific libraries to implement the token bucket algorithm for your workload. Check the examples section and do your own research to find a suitable library.
- Test limits that you plan to set, or that you plan to allow to be increased, and document the tested limits.
- Do not increase limits beyond what you establish in testing. When increasing a limit, verify that provisioned resources are already equivalent to or greater than those in test scenarios before applying the increase.

Resources

Related best practices:

- [REL04-BP03 Do constant work](#)
- [REL05-BP03 Control and limit retry calls](#)

Related documents:

- [Amazon API Gateway: Throttle API Requests for Better Throughput](#)

- [AWS WAF: Rate-based rule statement](#)
- [Introducing maximum concurrency of AWS Lambda when using Amazon SQS as an event source](#)
- [AWS Lambda: Maximum Concurrency](#)

Related examples:

- [The three most important AWS WAF rate-based rules](#)
- [Java Bucket4j](#)
- [Python token-bucket](#)
- [Node token-bucket](#)
- [.NET System Threading Rate Limiting](#)

Related videos:

- [Implementing GraphQL API security best practices with AWS AppSync](#)

Related tools:

- [Amazon API Gateway](#)
- [AWS AppSync](#)
- [Amazon SQS](#)
- [Amazon Kinesis](#)
- [AWS WAF](#)

REL05-BP03 Control and limit retry calls

Use exponential backoff to retry requests at progressively longer intervals between each retry. Introduce jitter between retries to randomize retry intervals. Limit the maximum number of retries.

Desired outcome: Typical components in a distributed software system include servers, load balancers, databases, and DNS servers. During normal operation, these components can respond to requests with errors that are temporary or limited, and also errors that would be persistent regardless of retries. When clients make requests to services, the requests consume resources including memory, threads, connections, ports, or any other limited resources. Controlling and

limiting retries is a strategy to release and minimize consumption of resources so that system components under strain are not overwhelmed.

When client requests time out or receive error responses, they should determine whether or not to retry. If they do retry, they do so with exponential backoff with jitter and a maximum retry value. As a result, backend services and processes are given relief from load and time to self-heal, resulting in faster recovery and successful request servicing.

Common anti-patterns:

- Implementing retries without adding exponential backoff, jitter, and maximum retry values. Backoff and jitter help avoid artificial traffic spikes due to unintentionally coordinated retries at common intervals.
- Implementing retries without testing their effects or assuming retries are already built into an SDK without testing retry scenarios.
- Failing to understand published error codes from dependencies, leading to retrying all errors, including those with a clear cause that indicates lack of permission, configuration error, or another condition that predictably will not resolve without manual intervention.
- Not addressing observability practices, including monitoring and alerting on repeated service failures so that underlying issues are made known and can be addressed.
- Developing custom retry mechanisms when built-in or third-party retry capabilities suffice.
- Retrying at multiple layers of your application stack in a manner which compounds retry attempts further consuming resources in a retry storm. Be sure to understand how these errors affect your application the dependencies you rely on, then implement retries at only one level.
- Retrying service calls that are not idempotent, causing unexpected side effects like duplicated results.

Benefits of establishing this best practice: Retries help clients acquire desired results when requests fail but also consume more of a server's time to get the successful responses they want. When failures are rare or transient, retries work well. When failures are caused by resource overload, retries can make things worse. Adding exponential backoff with jitter to client retries allows servers to recover when failures are caused by resource overload. Jitter avoids alignment of requests into spikes, and backoff diminishes load escalation caused by adding retries to normal request load. Finally, it's important to configure a maximum number of retries or elapsed time to avoid creating backlogs that produce metastable failures.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Control and limit retry calls. Use exponential backoff to retry after progressively longer intervals. Introduce jitter to randomize retry intervals and limit the maximum number of retries.

Some AWS SDKs implement retries and exponential backoff by default. Use these built-in AWS implementations where applicable in your workload. Implement similar logic in your workload when calling services that are idempotent and where retries improve your client availability. Decide what the timeouts are and when to stop retrying based on your use case. Build and exercise testing scenarios for those retry use cases.

Implementation steps

- Determine the optimal layer in your application stack to implement retries for the services your application relies on.
- Be aware of existing SDKs that implement proven retry strategies with exponential backoff and jitter for your language of choice, and favor these over writing your own retry implementations.
- Verify that [services are idempotent](#) before implementing retries. Once retries are implemented, be sure they are both tested and regularly exercise in production.
- When calling AWS service APIs, use the [AWS SDKs](#) and [AWS CLI](#) and understand the retry configuration options. Determine if the defaults work for your use case, test, and adjust as needed.

Resources

Related best practices:

- [REL04-BP04 Make all responses idempotent](#)
- [REL05-BP02 Throttle requests](#)
- [REL05-BP04 Fail fast and limit queues](#)
- [REL05-BP05 Set client timeouts](#)
- [REL11-BP01 Monitor all components of the workload to detect failures](#)

Related documents:

- [Error Retries and Exponential Backoff in AWS](#)
- [The Amazon Builders' Library: Timeouts, retries, and backoff with jitter](#)

- [Exponential Backoff and Jitter](#)
- [Making retries safe with idempotent APIs](#)

Related examples:

- [Spring Retry](#)
- [Resilience4j Retry](#)

Related videos:

- [Retry, backoff, and jitter: AWS re:Invent 2019: Introducing The Amazon Builders' Library \(DOP328\)](#)

Related tools:

- [AWS SDKs and Tools: Retry behavior](#)
- [AWS Command Line Interface: AWS CLI retries](#)

REL05-BP04 Fail fast and limit queues

When a service is unable to respond successfully to a request, fail fast. This allows resources associated with a request to be released, and permits a service to recover if it's running out of resources. Failing fast is a well-established software design pattern that can be leveraged to build highly reliable workloads in the cloud. Queuing is also a well-established enterprise integration pattern that can smooth load and allow clients to release resources when asynchronous processing can be tolerated. When a service is able to respond successfully under normal conditions but fails when the rate of requests is too high, use a queue to buffer requests. However, do not allow a buildup of long queue backlogs that can result in processing stale requests that a client has already given up on.

Desired outcome: When systems experience resource contention, timeouts, exceptions, or grey failures that make service level objectives unachievable, fail fast strategies allow for faster system recovery. Systems that must absorb traffic spikes and can accommodate asynchronous processing can improve reliability by allowing clients to quickly release requests by using queues to buffer requests to backend services. When buffering requests to queues, queue management strategies are implemented to avoid insurmountable backlogs.

Common anti-patterns:

- Implementing message queues but not configuring dead letter queues (DLQ) or alarms on DLQ volumes to detect when a system is in failure.
- Not measuring the age of messages in a queue, a measurement of latency to understand when queue consumers are falling behind or erroring out causing retrying.
- Not clearing backlogged messages from a queue, when there is no value in processing these messages if the business need no longer exists.
- Configuring first in first out (FIFO) queues when last in first out (LIFO) queues would better serve client needs, for example when strict ordering is not required and backlog processing is delaying all new and time sensitive requests resulting in all clients experiencing breached service levels.
- Exposing internal queues to clients instead of exposing APIs that manage work intake and place requests into internal queues.
- Combining too many work request types into a single queue which can exacerbate backlog conditions by spreading resource demand across request types.
- Processing complex and simple requests in the same queue, despite needing different monitoring, timeouts and resource allocations.
- Not validating inputs or using assertions to implement fail fast mechanisms in software that bubble up exceptions to higher level components that can handle errors gracefully.
- Not removing faulty resources from request routing, especially when failures are grey emitting both successes and failures due to crashing and restarting, intermittent dependency failure, reduced capacity, or network packet loss.

Benefits of establishing this best practice: Systems that fail fast are easier to debug and fix, and often expose issues in coding and configuration before releases are published into production. Systems that incorporate effective queueing strategies provide greater resilience and reliability to traffic spikes and intermittent system fault conditions.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Fail fast strategies can be coded into software solutions as well as configured into infrastructure. In addition to failing fast, queues are a straightforward yet powerful architectural technique to decouple system components smooth load. [Amazon CloudWatch](#) provides capabilities to monitor for and alarm on failures. Once a system is known to be failing, mitigation strategies can be

invoked, including failing away from impaired resources. When systems implement queues with [Amazon SQS](#) and other queue technologies to smooth load, they must consider how to manage queue backlogs, as well as message consumption failures.

Implementation steps

- Implement programmatic assertions or specific metrics in your software and use them to explicitly alert on system issues. Amazon CloudWatch helps you create metrics and alarms based on application log pattern and SDK instrumentation.
- Use CloudWatch metrics and alarms to fail away from impaired resources that are adding latency to processing or repeatedly failing to process requests.
- Use asynchronous processing by designing APIs to accept requests and append requests to internal queues using Amazon SQS and then respond to the message-producing client with a success message so the client can release resources and move on with other work while backend queue consumers process requests.
- Measure and monitor for queue processing latency by producing a CloudWatch metric each time you take a message off a queue by comparing now to message timestamp.
- When failures prevent successful message processing or traffic spikes in volumes that cannot be processed within service level agreements, sideline older or excess traffic to a spillover queue. This allows priority processing of new work, and older work when capacity is available. This technique is an approximation of LIFO processing and allows normal system processing for all new work.
- Use dead letter or redrive queues to move messages that can't be processed out of the backlog into a location that can be researched and resolved later
- Either retry or, when tolerable, drop old messages by comparing now to the message timestamp and discarding messages that are no longer relevant to the requesting client.

Resources

Related best practices:

- [REL04-BP02 Implement loosely coupled dependencies](#)
- [REL05-BP02 Throttle requests](#)
- [REL05-BP03 Control and limit retry calls](#)
- [REL06-BP02 Define and calculate metrics \(Aggregation\)](#)
- [REL06-BP07 Monitor end-to-end tracing of requests through your system](#)

Related documents:

- [Avoiding insurmountable queue backlogs](#)
- [Fail Fast](#)
- [How can I prevent an increasing backlog of messages in my Amazon SQS queue?](#)
- [Elastic Load Balancing: Zonal Shift](#)
- [Amazon Route 53 Application Recovery Controller: Routing control for traffic failover](#)

Related examples:

- [Enterprise Integration Patterns: Dead Letter Channel](#)

Related videos:

- [AWS re:Invent 2022 - Operating highly available Multi-AZ applications](#)

Related tools:

- [Amazon SQS](#)
- [Amazon MQ](#)
- [AWS IoT Core](#)
- [Amazon CloudWatch](#)

REL05-BP05 Set client timeouts

Set timeouts appropriately on connections and requests, verify them systematically, and do not rely on default values as they are not aware of workload specifics.

Desired outcome: Client timeouts should consider the cost to the client, server, and workload associated with waiting for requests that take abnormal amounts of time to complete. Since it is not possible to know the exact cause of any timeout, clients must use knowledge of services to develop expectations of probable causes and appropriate timeouts

Client connections time out based on configured values. After encountering a timeout, clients make decisions to back off and retry or open a [circuit breaker](#). These patterns avoid issuing requests that may exacerbate an underlying error condition.

Common anti-patterns:

- Not being aware of system timeouts or default timeouts.
- Not being aware of normal request completion timing.
- Not being aware of possible causes for requests to take abnormally long to complete, or the costs to client, service, or workload performance associated with waiting on these completions.
- Not being aware of the probability of impaired network causing a request to fail only once timeout is reached, and the costs to client and workload performance for not adopting a shorter timeout.
- Not testing timeout scenarios both for connections and requests.
- Setting timeouts too high, which can result in long wait times and increase resource utilization.
- Setting timeouts too low, resulting in artificial failures.
- Overlooking patterns to deal with timeout errors for remote calls like circuit breakers and retries.
- Not considering monitoring for service call error rates, service level objectives for latency, and latency outliers. These metrics can provide insight to aggressive or permissive timeouts

Benefits of establishing this best practice: Remote call timeouts are configured and systems are designed to handle timeouts gracefully so that resources are conserved when remote calls respond abnormally slow and timeout errors are handled gracefully by service clients.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Set both a connection timeout and a request timeout on any service dependency call and generally on any call across processes. Many frameworks offer built-in timeout capabilities, but be careful, as some have default values that are infinite or higher than acceptable for your service goals. A value that is too high reduces the usefulness of the timeout because resources continue to be consumed while the client waits for the timeout to occur. A value that is too low can generate increased traffic on the backend and increased latency because too many requests are retried. In some cases, this can lead to complete outages because all requests are being retried.

Consider the following when determining timeout strategies:

- Requests may take longer than normal to process because of their content, impairments in a target service, or a networking partition failure.

- Requests with abnormally expensive content could consume unnecessary server and client resources. In this case, timing out these requests and not retrying can preserve resources. Services should also protect themselves from abnormally expensive content with throttles and server-side timeouts.
- Requests that take abnormally long due to a service impairment can be timed out and retried. Consideration should be given to service costs for the request and retry, but if the cause is a localized impairment, a retry is not likely to be expensive and will reduce client resource consumption. The timeout may also release server resources depending on the nature of the impairment.
- Requests that take a long time to complete because the request or response has failed to be delivered by the network can be timed out and retried. Because the request or response was not delivered, failure would have been the outcome regardless of the length of timeout. Timing out in this case will not release server resources, but it will release client resources and improve workload performance.

Take advantage of well-established design patterns like retries and circuit breakers to handle timeouts gracefully and support fail-fast approaches. [AWS SDKs](#) and [AWS CLI](#) allow for configuration of both connection and request timeouts and for retries with exponential backoff and jitter. [AWS Lambda](#) functions support configuration of timeouts, and with [AWS Step Functions](#), you can build low code circuit breakers that take advantage of pre-built integrations with AWS services and SDKs. [AWS App Mesh](#) Envoy provides timeout and circuit breaker capabilities.

Implementation steps

- Configure timeouts on remote service calls and take advantage of built-in language timeout features or open source timeout libraries.
- When your workload makes calls with an AWS SDK, review the documentation for language specific timeout configuration.
 - [Python](#)
 - [PHP](#)
 - [.NET](#)
 - [Ruby](#)
 - [Java](#)
 - [Go](#)
 - [Node.js](#)

- [C++](#)
- When using AWS SDKs or AWS CLI commands in your workload, configure default timeout values by setting the AWS [configuration defaults](#) for `connectTimeoutInMillis` and `tlsNegotiationTimeoutInMillis`.
- Apply [command line options](#) `cli-connect-timeout` and `cli-read-timeout` to control one-off AWS CLI commands to AWS services.
- Monitor remote service calls for timeouts, and set alarms on persistent errors so that you can proactively handle error scenarios.
- Implement [CloudWatch Metrics](#) and [CloudWatch anomaly detection](#) on call error rates, service level objectives for latency, and latency outliers to provide insight into managing overly aggressive or permissive timeouts.
- Configure timeouts on [Lambda functions](#).
- API Gateway clients must implement their own retries when handling timeouts. API Gateway supports a [50 millisecond to 29 second integration timeout](#) for downstream integrations and does not retry when integration requests timeout.
- Implement the [circuit breaker](#) pattern to avoid making remote calls when they are timing out. Open the circuit to avoid failing calls and close the circuit when calls are responding normally.
- For container based workloads, review [App Mesh Envoy](#) features to leverage built in timeouts and circuit breakers.
- Use AWS Step Functions to build low code circuit breakers for remote service calls, especially where calling AWS native SDKs and supported Step Functions integrations to simplify your workload.

Resources

Related best practices:

- [REL05-BP03 Control and limit retry calls](#)
- [REL05-BP04 Fail fast and limit queues](#)
- [REL06-BP07 Monitor end-to-end tracing of requests through your system](#)

Related documents:

- [AWS SDK: Retries and Timeouts](#)

- [The Amazon Builders' Library: Timeouts, retries, and backoff with jitter](#)
- [Amazon API Gateway quotas and important notes](#)
- [AWS Command Line Interface: Command line options](#)
- [AWS SDK for Java 2.x: Configure API Timeouts](#)
- [AWS Botocore using the config object and Config Reference](#)
- [AWS SDK for .NET: Retries and Timeouts](#)
- [AWS Lambda: Configuring Lambda function options](#)

Related examples:

- [Using the circuit breaker pattern with AWS Step Functions and Amazon DynamoDB](#)
- [Martin Fowler: CircuitBreaker](#)

Related tools:

- [AWS SDKs](#)
- [AWS Lambda](#)
- [Amazon SQS](#)
- [AWS Step Functions](#)
- [AWS Command Line Interface](#)

REL05-BP06 Make services stateless where possible

Services should either not require state, or should offload state such that between different client requests, there is no dependence on locally stored data on disk and in memory. This allows servers to be replaced at will without causing an availability impact. Amazon ElastiCache or Amazon DynamoDB are good destinations for offloaded state.



Figure 7: In this stateless web application, session state is offloaded to Amazon ElastiCache.

When users or services interact with an application, they often perform a series of interactions that form a session. A session is unique data for users that persists between requests while they use the application. A stateless application is an application that does not need knowledge of previous interactions and does not store session information.

Once designed to be stateless, you can then use serverless compute services, such as AWS Lambda or AWS Fargate.

In addition to server replacement, another benefit of stateless applications is that they can scale horizontally because any of the available compute resources (such as EC2 instances and AWS Lambda functions) can service any request.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Make your applications stateless. Stateless applications allow horizontal scaling and are tolerant to the failure of an individual node.
 - Remove state that could actually be stored in request parameters.
 - After examining whether the state is required, move any state tracking to a resilient multi-zone cache or data store like Amazon ElastiCache, Amazon RDS, Amazon DynamoDB, or a third-party distributed data solution. Store a state that could not be moved to resilient data stores.
 - Some data (like cookies) can be passed in headers or query parameters.
 - Refactor to remove state that can be quickly passed in requests.
 - Some data may not actually be needed per request and can be retrieved on demand.
 - Remove data that can be asynchronously retrieved.
 - Decide on a data store that meets the requirements for a required state.
 - Consider a NoSQL database for non-relational data.

Resources

Related documents:

- [The Amazon Builders' Library: Avoiding fallback in distributed systems](#)
- [The Amazon Builders' Library: Avoiding insurmountable queue backlogs](#)
- [The Amazon Builders' Library: Caching challenges and strategies](#)

REL05-BP07 Implement emergency levers

This best practice was updated with new guidance on December 6, 2023.

Emergency levers are rapid processes that can mitigate availability impact on your workload.

Emergency levers work by disabling, throttling, or changing the behavior of components or dependencies using known and tested mechanisms. This can alleviate workload impairments caused by resource exhaustion due to unexpected increases in demand and reduce the impact of failures in non-critical components within your workload.

Desired outcome: By implementing emergency levers, you can establish known-good processes to maintain the availability of critical components in your workload. The workload should degrade gracefully and continue to perform its business-critical functions during the activation of an emergency lever. For more detail on graceful degradation, see [REL05-BP01 Implement graceful degradation to transform applicable hard dependencies into soft dependencies](#).

Common anti-patterns:

- Failure of non-critical dependencies impacts the availability of your core workload.
- Not testing or verifying critical component behavior during non-critical component impairment.
- No clear and deterministic criteria defined for activation or deactivation of an emergency lever.

Benefits of establishing this best practice: Implementing emergency levers can improve the availability of the critical components in your workload by providing your resolvers with established processes to respond to unexpected spikes in demand or failures of non-critical dependencies.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Identify critical components in your workload.
- Design and architect the critical components in your workload to withstand failure of non-critical components.
- Conduct testing to validate the behavior of your critical components during the failure of non-critical components.
- Define and monitor relevant metrics or triggers to initiate emergency lever procedures.
- Define the procedures (manual or automated) that comprise the emergency lever.

Implementation steps

- Identify business-critical components in your workload.
 - Each technical component in your workload should be mapped to its relevant business function and ranked as critical or non-critical. For examples of critical and non-critical functionality at Amazon, see [Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second](#).

- This is both a technical and business decision, and varies by organization and workload.
- Design and architect the critical components in your workload to withstand failure of non-critical components.
 - During dependency analysis, consider all potential failure modes, and verify that your emergency lever mechanisms deliver the critical functionality to downstream components.
- Conduct testing to validate the behavior of your critical components during activation of your emergency levers.
 - Avoid bimodal behavior. For more detail, see [REL11-BP05 Use static stability to prevent bimodal behavior](#).
- Define, monitor, and alert on relevant metrics to initiate the emergency lever procedure.
 - Finding the right metrics to monitor depends on your workload. Some example metrics are latency or the number of failed request to a dependency.
- Define the procedures, manual or automated, that comprise the emergency lever.
 - This may include mechanisms such as [load shedding](#), [throttling requests](#), or implementing [graceful degradation](#).

Resources

Related best practices:

- [REL05-BP01 Implement graceful degradation to transform applicable hard dependencies into soft dependencies](#)
- [REL05-BP02 Throttle requests](#)
- [REL11-BP05 Use static stability to prevent bimodal behavior](#)

Related documents:

- [Automating safe, hands-off deployments](#)
- [Any Day Can Be Prime Day: How Amazon.com Search Uses Chaos Engineering to Handle Over 84K Requests Per Second](#)

Related videos:

- [AWS re:Invent 2020: Reliability, consistency, and confidence through immutability](#)

Change management

Questions

- [REL 6. How do you monitor workload resources?](#)
- [REL 7. How do you design your workload to adapt to changes in demand?](#)
- [REL 8. How do you implement change?](#)

REL 6. How do you monitor workload resources?

Logs and metrics are powerful tools to gain insight into the health of your workload. You can configure your workload to monitor logs and metrics and send notifications when thresholds are crossed or significant events occur. Monitoring allows your workload to recognize when low-performance thresholds are crossed or failures occur, so it can recover automatically in response.

Best practices

- [REL06-BP01 Monitor all components for the workload \(Generation\)](#)
- [REL06-BP02 Define and calculate metrics \(Aggregation\)](#)
- [REL06-BP03 Send notifications \(Real-time processing and alarming\)](#)
- [REL06-BP04 Automate responses \(Real-time processing and alarming\)](#)
- [REL06-BP05 Analytics](#)
- [REL06-BP06 Conduct reviews regularly](#)
- [REL06-BP07 Monitor end-to-end tracing of requests through your system](#)

REL06-BP01 Monitor all components for the workload (Generation)

Monitor the components of the workload with Amazon CloudWatch or third-party tools. Monitor AWS services with AWS Health Dashboard.

All components of your workload should be monitored, including the front-end, business logic, and storage tiers. Define key metrics, describe how to extract them from logs (if necessary), and set thresholds for invoking corresponding alarm events. Ensure metrics are relevant to the key performance indicators (KPIs) of your workload, and use metrics and logs to identify early warning signs of service degradation. For example, a metric related to business outcomes such as the number of orders successfully processed per minute, can indicate workload issues faster than

technical metric, such as CPU Utilization. Use AWS Health Dashboard for a personalized view into the performance and availability of the AWS services underlying your AWS resources.

Monitoring in the cloud offers new opportunities. Most cloud providers have developed customizable hooks and can deliver insights to help you monitor multiple layers of your workload. AWS services such as Amazon CloudWatch apply statistical and machine learning algorithms to continually analyze metrics of systems and applications, determine normal baselines, and surface anomalies with minimal user intervention. Anomaly detection algorithms account for the seasonality and trend changes of metrics.

AWS makes an abundance of monitoring and log information available for consumption that can be used to define workload-specific metrics, change-in-demand processes, and adopt machine learning techniques regardless of ML expertise.

In addition, monitor all of your external endpoints to ensure that they are independent of your base implementation. This active monitoring can be done with synthetic transactions (sometimes referred to as *user canaries*, but not to be confused with canary deployments) which periodically run a number of common tasks matching actions performed by clients of the workload. Keep these tasks short in duration and be sure not to overload your workload during testing. Amazon CloudWatch Synthetics allows you to [create synthetic canaries](#) to monitor your endpoints and APIs. You can also combine the synthetic canary client nodes with AWS X-Ray console to pinpoint which synthetic canaries are experiencing issues with errors, faults, or throttling rates for the selected time frame.

Desired Outcome:

Collect and use critical metrics from all components of the workload to ensure workload reliability and optimal user experience. Detecting that a workload is not achieving business outcomes allows you to quickly declare a disaster and recover from an incident.

Common anti-patterns:

- Only monitoring external interfaces to your workload.
- Not generating any workload-specific metrics and only relying on metrics provided to you by the AWS services your workload uses.
- Only using technical metrics in your workload and not monitoring any metrics related to non-technical KPIs the workload contributes to.
- Relying on production traffic and simple health checks to monitor and evaluate workload state.

Benefits of establishing this best practice: Monitoring at all tiers in your workload allows you to more rapidly anticipate and resolve problems in the components that comprise the workload.

Level of risk exposed if this best practice is not established: High

Implementation guidance

1. **Turn on logging where available.** Monitoring data should be obtained from all components of the workloads. Turn on additional logging, such as S3 Access Logs, and permit your workload to log workload specific data. Collect metrics for CPU, network I/O, and disk I/O averages from services such as Amazon ECS, Amazon EKS, Amazon EC2, Elastic Load Balancing, AWS Auto Scaling, and Amazon EMR. See [AWS Services That Publish CloudWatch Metrics](#) for a list of AWS services that publish metrics to CloudWatch.
2. **Review all default metrics and explore any data collection gaps.** Every service generates default metrics. Collecting default metrics allows you to better understand the dependencies between workload components, and how component reliability and performance affect the workload. You can also create and [publish your own metrics](#) to CloudWatch using the AWS CLI or an API.
3. **Evaluate all the metrics to decide which ones to alert on for each AWS service in your workload.** You may choose to select a subset of metrics that have a major impact on workload reliability. Focusing on critical metrics and threshold allows you to refine the number of [alerts](#) and can help minimize false-positives.
4. **Define alerts and the recovery process for your workload after the alert is invoked.** Defining alerts allows you to quickly notify, escalate, and follow steps necessary to recover from an incident and meet your prescribed Recovery Time Objective (RTO). You can use [Amazon CloudWatch Alarms](#) to invoke automated workflows and initiate recovery procedures based on defined thresholds.
5. **Explore use of synthetic transactions to collect relevant data about workloads state.** Synthetic monitoring follows the same routes and perform the same actions as a customer, which makes it possible for you to continually verify your customer experience even when you don't have any customer traffic on your workloads. By using [synthetic transactions](#), you can discover issues before your customers do.

Resources

Related best practices:

- [REL11-BP03 Automate healing on all layers](#)

Related documents:

- [Getting started with your AWS Health Dashboard – Your account health](#)
- [AWS Services That Publish CloudWatch Metrics](#)
- [Access Logs for Your Network Load Balancer](#)
- [Access logs for your application load balancer](#)
- [Accessing Amazon CloudWatch Logs for AWS Lambda](#)
- [Amazon S3 Server Access Logging](#)
- [Enable Access Logs for Your Classic Load Balancer](#)
- [Exporting log data to Amazon S3](#)
- [Install the CloudWatch agent on an Amazon EC2 instance](#)
- [Publishing Custom Metrics](#)
- [Using Amazon CloudWatch Dashboards](#)
- [Using Amazon CloudWatch Metrics](#)
- [Using Canaries \(Amazon CloudWatch Synthetics\)](#)
- [What are Amazon CloudWatch Logs?](#)

User guides:

- [Creating a trail](#)
- [Monitoring memory and disk metrics for Amazon EC2 Linux instances](#)
- [Using CloudWatch Logs with container instances](#)
- [VPC Flow Logs](#)
- [What is Amazon DevOps Guru?](#)
- [What is AWS X-Ray?](#)

Related blogs:

- [Debugging with Amazon CloudWatch Synthetics and AWS X-Ray](#)

Related examples and workshops:

- [AWS Well-Architected Labs: Operational Excellence - Dependency Monitoring](#)
- [The Amazon Builders' Library: Instrumenting distributed systems for operational visibility](#)
- [Observability workshop](#)

REL06-BP02 Define and calculate metrics (Aggregation)

Store log data and apply filters where necessary to calculate metrics, such as counts of a specific log event, or latency calculated from log event timestamps.

Amazon CloudWatch and Amazon S3 serve as the primary aggregation and storage layers. For some services, such as AWS Auto Scaling and Elastic Load Balancing, default metrics are provided by default for CPU load or average request latency across a cluster or instance. For streaming services, such as VPC Flow Logs and AWS CloudTrail, event data is forwarded to CloudWatch Logs and you need to define and apply metrics filters to extract metrics from the event data. This gives you time series data, which can serve as inputs to CloudWatch alarms that you define to invoke alerts.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Define and calculate metrics (Aggregation). Store log data and apply filters where necessary to calculate metrics, such as counts of a specific log event, or latency calculated from log event timestamps
 - Metric filters define the terms and patterns to look for in log data as it is sent to CloudWatch Logs. CloudWatch Logs uses these metric filters to turn log data into numerical CloudWatch metrics that you can graph or set an alarm on.
 - [Searching and Filtering Log Data](#)
 - Use a trusted third party to aggregate logs.
 - Follow the instructions of the third party. Most third-party products integrate with CloudWatch and Amazon S3.
 - Some AWS services can publish logs directly to Amazon S3. If your main requirement for logs is storage in Amazon S3, you can easily have the service producing the logs send them directly to Amazon S3 without setting up additional infrastructure.
 - [Sending Logs Directly to Amazon S3](#)

Resources

Related documents:

- [Amazon CloudWatch Logs Insights Sample Queries](#)
- [Debugging with Amazon CloudWatch Synthetics and AWS X-Ray](#)
- [One Observability Workshop](#)
- [Searching and Filtering Log Data](#)
- [Sending Logs Directly to Amazon S3](#)
- [The Amazon Builders' Library: Instrumenting distributed systems for operational visibility](#)

REL06-BP03 Send notifications (Real-time processing and alarming)

When organizations detect potential issues, they send real-time notifications and alerts to the appropriate personnel and systems in order to respond quickly and effectively to these issues.

Desired outcome: Rapid responses to operational events are possible through configuration of relevant alarms based on service and application metrics. When alarm thresholds are breached, the appropriate personnel and systems are notified so they can address underlying issues.

Common anti-patterns:

- Configuring alarms with an excessively high threshold, resulting in the failure to send vital notifications.
- Configuring alarms with a threshold that is too low, resulting in inaction on important alerts due to the noise of excessive notifications.
- Not updating alarms and their threshold when usage changes.
- For alarms best addressed through automated actions, sending the notification to personnel instead of generating the automated action results in excessive notifications being sent.

Benefits of establishing this best practice: Sending real-time notifications and alerts to the appropriate personnel and systems allows for early detection of issues and rapid responses to operational incidents.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Workloads should be equipped with real-time processing and alarming to improve the detectability of issues that could impact the availability of the application and serve as triggers for automated response. Organizations can perform real-time processing and alarming by creating alerts with defined metrics in order to receive notifications whenever significant events occur or a metric exceeds a threshold.

[Amazon CloudWatch](#) allows you to create [metric](#) and composite alarms using CloudWatch alarms based on static threshold, anomaly detection, and other criteria. For more detail on the types of alarms you can configure using CloudWatch, see the [alarms section of the CloudWatch documentation](#).

You can construct customized views of metrics and alerts of your AWS resources for your teams using [CloudWatch dashboards](#). The customizable home pages in the CloudWatch console allow you to monitor your resources in a single view across multiple Regions.

Alarms can perform one or more actions, like sending a notification to an [Amazon SNS topic](#), performing an [Amazon EC2](#) action or an [Amazon EC2 Auto Scaling](#) action, or [creating an OpsItem](#) or [incident](#) in AWS Systems Manager.

Amazon CloudWatch uses [Amazon SNS](#) to send notifications when the alarm changes state, providing message delivery from the publishers (producers) to the subscribers (consumers). For more detail on setting up Amazon SNS notifications, see [Configuring Amazon SNS](#).

CloudWatch sends [EventBridge events](#) whenever a CloudWatch alarm is created, updated, deleted, or its state changes. You can use EventBridge with these events to create rules that perform actions, such as notifying you whenever the state of an alarm changes or automatically triggering events in your account using [Systems Manager automation](#).

When should you use EventBridge or Amazon SNS?

Both EventBridge and Amazon SNS can be used to develop event-driven applications, and your choice will depend on your specific needs.

Amazon EventBridge is recommended when you want to build an application that reacts to events from your own applications, SaaS applications, and AWS services. EventBridge is the only event-based service that integrates directly with third-party SaaS partners. EventBridge also automatically ingests events from over 200 AWS services without requiring developers to create any resources in their account.

EventBridge uses a defined JSON-based structure for events, and helps you create rules that are applied across the entire event body to select events to forward to a [target](#). EventBridge currently supports over 20 AWS services as targets, including [AWS Lambda](#), [Amazon SQS](#), [Amazon SNS](#), [Amazon Kinesis Data Streams](#), and [Amazon Data Firehose](#).

Amazon SNS is recommended for applications that need high fan out (thousands or millions of endpoints). A common pattern we see is that customers use Amazon SNS as a target for their rule to filter the events that they need, and fan out to multiple endpoints.

Messages are unstructured and can be in any format. Amazon SNS supports forwarding messages to six different types of targets, including Lambda, Amazon SQS, HTTP/S endpoints, SMS, mobile push, and email. Amazon SNS [typical latency is under 30 milliseconds](#). A wide range of AWS services send Amazon SNS messages by configuring the service to do so (more than 30, including Amazon EC2, [Amazon S3](#), and [Amazon RDS](#)).

Implementation steps

1. Create an alarm using [Amazon CloudWatch alarms](#).
 - a. A metric alarm monitors a single CloudWatch metric or an expression dependent on CloudWatch metrics. The alarm initiates one or more actions based on the value of the metric or expression in comparison to a threshold over a number of time intervals. The action may consist of sending a notification to an [Amazon SNS topic](#), performing an [Amazon EC2](#) action or an [Amazon EC2 Auto Scaling](#) action, or [creating an OpsItem](#) or [incident](#) in AWS Systems Manager.
 - b. A composite alarm consists of a rule expression that considers the alarm conditions of other alarms you've created. The composite alarm only enters alarm state if all rule conditions are met. The alarms specified in the rule expression of a composite alarm can include metric alarms and additional composite alarms. Composite alarms can send Amazon SNS notifications when their state changes and can create Systems Manager [OpsItems](#) or [incidents](#) when they enter the alarm state, but they cannot perform Amazon EC2 or Auto Scaling actions.
2. Set up [Amazon SNS notifications](#). When creating a CloudWatch alarm, you can include an Amazon SNS topic to send a notification when the alarm changes state.
3. [Create rules in EventBridge](#) that matches specified CloudWatch alarms. Each rule supports multiple targets, including Lambda functions. For example, you can define an alarm that initiates when available disk space is running low, which triggers a Lambda function through an

EventBridge rule, to clean up the space. For more detail on EventBridge targets, see [EventBridge targets](#).

Resources

Related Well-Architected best practices:

- [REL06-BP01 Monitor all components for the workload \(Generation\)](#)
- [REL06-BP02 Define and calculate metrics \(Aggregation\)](#)
- [REL12-BP01 Use playbooks to investigate failures](#)

Related documents:

- [Amazon CloudWatch](#)
- [CloudWatch Logs insights](#)
- [Using Amazon CloudWatch alarms](#)
- [Using Amazon CloudWatch dashboards](#)
- [Using Amazon CloudWatch metrics](#)
- [Setting up Amazon SNS notifications](#)
- [CloudWatch anomaly detection](#)
- [CloudWatch Logs data protection](#)
- [Amazon EventBridge](#)
- [Amazon Simple Notification Service](#)

Related videos:

- [reinvent 2022 observability videos](#)
- [AWS re:Invent 2022 - Observability best practices at Amazon](#)

Related examples:

- [One Observability Workshop](#)
- [Amazon EventBridge to AWS Lambda with feedback control by Amazon CloudWatch Alarms](#)

REL06-BP04 Automate responses (Real-time processing and alarming)

This best practice was updated with new guidance on December 6, 2023.

Use automation to take action when an event is detected, for example, to replace failed components.

Automated real-time processing of alarms is implemented so that systems can take quick corrective action and attempt to prevent failures or degraded service when alarms are triggered. Automated responses to alarms could include the replacement of failing components, the adjustment of compute capacity, the redirection of traffic to healthy hosts, availability zones, or other regions, and the notification of operators.

Desired outcome: Real-time alarms are identified, and automated processing of alarms is set up to invoke the appropriate actions taken to maintain service level objectives and service-level agreements (SLAs). Automation can range from self-healing activities of single components to full-site failover.

Common anti-patterns:

- Not having a clear inventory or catalog of key real-time alarms.
- No automated responses on critical alarms (for example, when compute is nearing exhaustion, autoscaling occurs).
- Contradictory alarm response actions.
- No standard operating procedures (SOPs) for operators to follow when they receive alert notifications.
- Not monitoring configuration changes, as undetected configuration changes can cause downtime for workloads.
- Not having a strategy to undo unintended configuration changes.

Benefits of establishing this best practice: Automating alarm processing can improve system resiliency. The system takes corrective actions automatically, reducing manual activities that allow for human, error-prone interventions. Workload operates meet availability goals, and reduces service disruption.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

To effectively manage alerts and automate their response, categorize alerts based on their criticality and impact, document response procedures, and plan responses before ranking tasks.

Identify tasks requiring specific actions (often detailed in runbooks), and examine all runbooks and playbooks to determine which tasks can be automated. If actions can be defined, often they can be automated. If actions cannot be automated, document manual steps in an SOP and train operators on them. Continually challenge manual processes for automation opportunities where you can establish and maintain a plan to automate alert responses.

Implementation steps

- 1. Create an inventory of alarms:** To obtain a list of all alarms, you can use the [AWS CLI](#) using the [Amazon CloudWatch](#) command [describe-alarms](#). Depending upon how many alarms you have set up, you might have to use pagination to retrieve a subset of alarms for each call, or alternatively you can use the AWS SDK to obtain the alarms [using an API call](#).
- 2. Document all alarm actions:** Update a runbook with all alarms and their actions, irrespective if they are manual or automated. [AWS Systems Manager](#) provides predefined runbooks. For more information about runbooks, see [Working with runbooks](#). For detail on how to view runbook content, see [View runbook content](#).
- 3. Set up and manage alarm actions:** For any of the alarms that require an action, specify the [automated action using the CloudWatch SDK](#). For example, you can change the state of your Amazon EC2 instances automatically based on a CloudWatch alarm by creating and enabling actions on an alarm or disabling actions on an alarm.

You can also use [Amazon EventBridge](#) to respond automatically to system events, such as application availability issues or resource changes. You can create rules to indicate which events you're interested in, and the actions to take when an event matches a rule. The actions that can be automatically initiated include invoking an [AWS Lambda](#) function, invoking [Amazon EC2 Run Command](#), relaying the event to [Amazon Kinesis Data Streams](#), and seeing [Automate Amazon EC2 using EventBridge](#).

- 4. Standard Operating Procedures (SOPs):** Based on your application components, [AWS Resilience Hub](#) recommends multiple [SOP templates](#). You can use these SOPs to document all the processes an operator should follow in case an alert is raised. You can also [construct a SOP](#) based on Resilience Hub recommendations, where you need an Resilience Hub application with an associated resiliency policy, as well as a historic resiliency assessment against that application. The recommendations for your SOP are produced by the resiliency assessment.

Resilience Hub works with Systems Manager to automate the steps of your SOPs by providing a number of [SSM documents](#) you can use as the basis for those SOPs. For example, Resilience Hub may recommend an SOP for adding disk space based on an existing SSM automation document.

5. Perform automated actions using Amazon DevOps Guru: You can use [Amazon DevOps Guru](#) to automatically monitor application resources for anomalous behavior and deliver targeted recommendations to speed up problem identification and remediation times. With DevOps Guru, you can monitor streams of operational data in near real time from multiple sources including Amazon CloudWatch metrics, [AWS Config](#), [AWS CloudFormation](#), and [AWS X-Ray](#). You can also use DevOps Guru to automatically create [OpsItems](#) in OpsCenter and send events to EventBridge for additional automation.

Resources

Related best practices:

- [REL06-BP01 Monitor all components for the workload \(Generation\)](#)
- [REL06-BP02 Define and calculate metrics \(Aggregation\)](#)
- [REL06-BP03 Send notifications \(Real-time processing and alarming\)](#)
- [REL08-BP01 Use runbooks for standard activities such as deployment](#)

Related documents:

- [AWS Systems Manager Automation](#)
- [Creating an EventBridge Rule That Triggers on an Event from an AWS Resource](#)
- [One Observability Workshop](#)
- [The Amazon Builders' Library: Instrumenting distributed systems for operational visibility](#)
- [What is Amazon DevOps Guru?](#)
- [Working with Automation Documents \(Playbooks\)](#)

Related videos:

- [AWS re:Invent 2022 - Observability best practices at Amazon](#)
- [AWS re:Invent 2020: Automate anything with AWS Systems Manager](#)
- [Introduction to AWS Resilience Hub](#)

- [Create Custom Ticket Systems for Amazon DevOps Guru Notifications](#)
- [Enable Multi-Account Insight Aggregation with Amazon DevOps Guru](#)

Related examples:

- [Reliability Workshops](#)
- [Amazon CloudWatch and Systems Manager Workshop](#)

REL06-BP05 Analytics

Collect log files and metrics histories and analyze these for broader trends and workload insights.

Amazon CloudWatch Logs Insights supports a [simple yet powerful query language](#) that you can use to analyze log data. Amazon CloudWatch Logs also supports subscriptions that allow data to flow seamlessly to Amazon S3 where you can use or Amazon Athena to query the data. It also supports queries on a large array of formats. See [Supported SerDes and Data Formats](#) in the Amazon Athena User Guide for more information. For analysis of huge log file sets, you can run an Amazon EMR cluster to run petabyte-scale analyses.

There are a number of tools provided by AWS Partners and third parties that allow for aggregation, processing, storage, and analytics. These tools include New Relic, Splunk, Loggly, Logstash, CloudHealth, and Nagios. However, outside generation of system and application logs is unique to each cloud provider, and often unique to each service.

An often-overlooked part of the monitoring process is data management. You need to determine the retention requirements for monitoring data, and then apply lifecycle policies accordingly. Amazon S3 supports lifecycle management at the S3 bucket level. This lifecycle management can be applied differently to different paths in the bucket. Toward the end of the lifecycle, you can transition data to Amazon S3 Glacier for long-term storage, and then expiration after the end of the retention period is reached. The S3 Intelligent-Tiering storage class is designed to optimize costs by automatically moving data to the most cost-effective access tier, without performance impact or operational overhead.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- CloudWatch Logs Insights allows you to interactively search and analyze your log data in Amazon CloudWatch Logs.

- [Analyzing Log Data with CloudWatch Logs Insights](#)
- [Amazon CloudWatch Logs Insights Sample Queries](#)
- Use Amazon CloudWatch Logs send logs to Amazon S3 where you can use or Amazon Athena to query the data.
 - [How do I analyze my Amazon S3 server access logs using Athena?](#)
 - Create an S3 lifecycle policy for your server access logs bucket. Configure the lifecycle policy to periodically remove log files. Doing so reduces the amount of data that Athena analyzes for each query.
 - [How Do I Create a Lifecycle Policy for an S3 Bucket?](#)

Resources

Related documents:

- [Amazon CloudWatch Logs Insights Sample Queries](#)
- [Analyzing Log Data with CloudWatch Logs Insights](#)
- [Debugging with Amazon CloudWatch Synthetics and AWS X-Ray](#)
- [How Do I Create a Lifecycle Policy for an S3 Bucket?](#)
- [How do I analyze my Amazon S3 server access logs using Athena?](#)
- [One Observability Workshop](#)
- [The Amazon Builders' Library: Instrumenting distributed systems for operational visibility](#)

REL06-BP06 Conduct reviews regularly

Frequently review how workload monitoring is implemented and update it based on significant events and changes.

Effective monitoring is driven by key business metrics. Ensure these metrics are accommodated in your workload as business priorities change.

Auditing your monitoring helps ensure that you know when an application is meeting its availability goals. Root cause analysis requires the ability to discover what happened when failures occur. AWS provides services that allow you to track the state of your services during an incident:

- **Amazon CloudWatch Logs:** You can store your logs in this service and inspect their contents.

- **Amazon CloudWatch Logs Insights:** Is a fully managed service that allows you to analyze massive logs in seconds. It gives you fast, interactive queries and visualizations.
- **AWS Config:** You can see what AWS infrastructure was in use at different points in time.
- **AWS CloudTrail:** You can see which AWS APIs were invoked at what time and by what principal.

At AWS, we conduct a weekly meeting to [review operational performance](#) and to share learnings between teams. Because there are so many teams in AWS, we created [The Wheel](#) to randomly pick a workload to review. Establishing a regular cadence for operational performance reviews and knowledge sharing enhances your ability to achieve higher performance from your operational teams.

Common anti-patterns:

- Collecting only default metrics.
- Setting a monitoring strategy and never reviewing it.
- Not discussing monitoring when major changes are deployed.

Benefits of establishing this best practice: Regularly reviewing your monitoring allows for the anticipation of potential problems, instead of reacting to notifications when an anticipated problem actually occurs.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Create multiple dashboards for the workload. You must have a top-level dashboard that contains the key business metrics, as well as the technical metrics you have identified to be the most relevant to the projected health of the workload as usage varies. You should also have dashboards for various application tiers and dependencies that can be inspected.
 - [Using Amazon CloudWatch Dashboards](#)
- Schedule and conduct regular reviews of the workload dashboards. Conduct regular inspection of the dashboards. You may have different cadences for the depth at which you inspect.
 - Inspect for trends in the metrics. Compare the metric values to historic values to see if there are trends that may indicate that something needs investigation. Examples of this include: increasing latency, decreasing primary business function, and increasing failure responses.

- Inspect for outliers/anomalies in your metrics. Averages or medians can mask outliers and anomalies. Look at the highest and lowest values during the time frame and investigate the causes of extreme scores. As you continue to eliminate these causes, lowering your definition of extreme allows you to continue to improve the consistency of your workload performance.
- Look for sharp changes in behavior. An immediate change in quantity or direction of a metric may indicate that there has been a change in the application, or external factors that you may need to add additional metrics to track.

Resources

Related documents:

- [Amazon CloudWatch Logs Insights Sample Queries](#)
- [Debugging with Amazon CloudWatch Synthetics and AWS X-Ray](#)
- [One Observability Workshop](#)
- [The Amazon Builders' Library: Instrumenting distributed systems for operational visibility](#)
- [Using Amazon CloudWatch Dashboards](#)

REL06-BP07 Monitor end-to-end tracing of requests through your system

Trace requests as they process through service components so product teams can more easily analyze and debug issues and improve performance.

Desired outcome: Workloads with comprehensive tracing across all components are easy to debug, improving [mean time to resolution](#) (MTTR) of errors and latency by simplifying root cause discovery. End-to-end tracing reduces the time it takes to discover impacted components and drill into the detailed root causes of errors or latency.

Common anti-patterns:

- Tracing is used for some components but not for all. For example, without tracing for AWS Lambda, teams might not clearly understand latency caused by cold starts in a spiky workload.
- Synthetic canaries or real-user monitoring (RUM) are not configured with tracing. Without canaries or RUM, client interaction telemetry is omitted from the trace analysis yielding an incomplete performance profile.
- Hybrid workloads include both cloud native and third party tracing tools, but steps have not been taken elect and fully integrate a single tracing solution. Based on the elected tracing

solution, cloud native tracing SDKs should be used to instrument components that are not cloud native or third party tools should be configured to ingest cloud native trace telemetry.

Benefits of establishing this best practice: When development teams are alerted to issues, they can see a full picture of system component interactions, including component by component correlation to logging, performance, and failures. Because tracing makes it easy to visually identify root causes, less time is spent investigating root causes. Teams that understand component interactions in detail make better and faster decisions when resolving issues. Decisions like when to invoke disaster recovery (DR) failover or where to best implement self-healing strategies can be improved by analyzing systems traces, ultimately improving customer satisfaction with your services.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Teams that operate distributed applications can use tracing tools to establish a correlation identifier, collect traces of requests, and build service maps of connected components. All application components should be included in request traces including service clients, middleware gateways and event buses, compute components, and storage, including key value stores and databases. Include synthetic canaries and real-user monitoring in your end-to-end tracing configuration to measure remote client interactions and latency so that you can accurately evaluate your systems performance against your service level agreements and objectives.

You can use [AWS X-Ray](#) and [Amazon CloudWatch Application Monitoring](#) instrumentation services to provide a complete view of requests as they travel through your application. X-Ray collects application telemetry and allows you to visualize and filter it across payloads, functions, traces, services, APIs, and can be turned on for system components with no-code or low-code. CloudWatch application monitoring includes ServiceLens to integrate your traces with metrics, logs, and alarms. CloudWatch application monitoring also includes synthetics to monitor your endpoints and APIs, as well as real-user monitoring to instrument your web application clients.

Implementation steps

- Use AWS X-Ray on all supported native services like [Amazon S3, AWS Lambda, and Amazon API Gateway](#). These AWS services enable X-Ray with configuration toggles using infrastructure as code, AWS SDKs, or the AWS Management Console.
- Instrument applications [AWS Distro for Open Telemetry and X-Ray](#) or third-party collection agents.

- Review the [AWS X-Ray Developer Guide](#) for programming language specific implementation. These documentation sections detail how to instrument HTTP requests, SQL queries, and other processes specific to your application programming language.
- Use X-Ray tracing for [Amazon CloudWatch Synthetic Canaries](#) and [Amazon CloudWatch RUM](#) to analyze the request path from your end user client through your downstream AWS infrastructure.
- Configure CloudWatch metrics and alarms based on resource health and canary telemetry so that teams are alerted to issues quickly, and can then deep dive into traces and service maps with ServiceLens.
- Enable X-Ray integration for third party tracing tools like [Datadog](#), [New Relic](#), or [Dynatrace](#) if you are using third party tools for your primary tracing solution.

Resources

Related best practices:

- [REL06-BP01 Monitor all components for the workload \(Generation\)](#)
- [REL11-BP01 Monitor all components of the workload to detect failures](#)

Related documents:

- [What is AWS X-Ray?](#)
- [Amazon CloudWatch: Application Monitoring](#)
- [Debugging with Amazon CloudWatch Synthetics and AWS X-Ray](#)
- [The Amazon Builders' Library: Instrumenting distributed systems for operational visibility](#)
- [Integrating AWS X-Ray with other AWS services](#)
- [AWS Distro for OpenTelemetry and AWS X-Ray](#)
- [Amazon CloudWatch: Using synthetic monitoring](#)
- [Amazon CloudWatch: Use CloudWatch RUM](#)
- [Set up Amazon CloudWatch synthetics canary and Amazon CloudWatch alarm](#)
- [Availability and Beyond: Understanding and Improving the Resilience of Distributed Systems on AWS](#)

Related examples:

- [One Observability Workshop](#)

Related videos:

- [AWS re:Invent 2022 - How to monitor applications across multiple accounts](#)
- [How to Monitor your AWS Applications](#)

Related tools:

- [AWS X-Ray](#)
- [Amazon CloudWatch](#)
- [Amazon Route 53](#)

REL 7. How do you design your workload to adapt to changes in demand?

A scalable workload provides elasticity to add or remove resources automatically so that they closely match the current demand at any given point in time.

Best practices

- [REL07-BP01 Use automation when obtaining or scaling resources](#)
- [REL07-BP02 Obtain resources upon detection of impairment to a workload](#)
- [REL07-BP03 Obtain resources upon detection that more resources are needed for a workload](#)
- [REL07-BP04 Load test your workload](#)

REL07-BP01 Use automation when obtaining or scaling resources

When replacing impaired resources or scaling your workload, automate the process by using managed AWS services, such as Amazon S3 and AWS Auto Scaling. You can also use third-party tools and AWS SDKs to automate scaling.

Managed AWS services include Amazon S3, Amazon CloudFront, AWS Auto Scaling, AWS Lambda, Amazon DynamoDB, AWS Fargate, and Amazon Route 53.

AWS Auto Scaling lets you detect and replace impaired instances. It also lets you build scaling plans for resources including [Amazon EC2](#) instances and Spot Fleets, [Amazon ECS](#) tasks, [Amazon DynamoDB](#) tables and indexes, and [Amazon Aurora](#) Replicas.

When scaling EC2 instances, ensure that you use multiple Availability Zones (preferably at least three) and add or remove capacity to maintain balance across these Availability Zones. ECS tasks or Kubernetes pods (when using Amazon Elastic Kubernetes Service) should also be distributed across multiple Availability Zones.

When using AWS Lambda, instances scale automatically. Every time an event notification is received for your function, AWS Lambda quickly locates free capacity within its compute fleet and runs your code up to the allocated concurrency. You need to ensure that the necessary concurrency is configured on the specific Lambda, and in your Service Quotas.

Amazon S3 automatically scales to handle high request rates. For example, your application can achieve at least 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second per prefix in a bucket. There are no limits to the number of prefixes in a bucket. You can increase your read or write performance by parallelizing reads. For example, if you create 10 prefixes in an Amazon S3 bucket to parallelize reads, you could scale your read performance to 55,000 read requests per second.

Configure and use Amazon CloudFront or a trusted content delivery network (CDN). A CDN can provide faster end-user response times and can serve requests for content from cache, therefore reducing the need to scale your workload.

Common anti-patterns:

- Implementing Auto Scaling groups for automated healing, but not implementing elasticity.
- Using automatic scaling to respond to large increases in traffic.
- Deploying highly stateful applications, eliminating the option of elasticity.

Benefits of establishing this best practice: Automation removes the potential for manual error in deploying and decommissioning resources. Automation removes the risk of cost overruns and denial of service due to slow response on needs for deployment or decommissioning.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Configure and use AWS Auto Scaling. This monitors your applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost. Using AWS Auto Scaling, you can setup application scaling for multiple resources across multiple services.

- [What is AWS Auto Scaling?](#)

- Configure Auto Scaling on your Amazon EC2 instances and Spot Fleets, Amazon ECS tasks, Amazon DynamoDB tables and indexes, Amazon Aurora Replicas, and AWS Marketplace appliances as applicable.

- [Managing throughput capacity automatically with DynamoDB Auto Scaling](#)

- Use service API operations to specify the alarms, scaling policies, warm up times, and cool down times.

- Use Elastic Load Balancing. Load balancers can distribute load by path or by network connectivity.

- [What is Elastic Load Balancing?](#)

- Application Load Balancers can distribute load by path.

- [What is an Application Load Balancer?](#)

- Configure an Application Load Balancer to distribute traffic to different workloads based on the path under the domain name.

- Application Load Balancers can be used to distribute loads in a manner that integrates with AWS Auto Scaling to manage demand.

- [Using a load balancer with an Auto Scaling group](#)

- Network Load Balancers can distribute load by connection.

- [What is a Network Load Balancer?](#)

- Configure a Network Load Balancer to distribute traffic to different workloads using TCP, or to have a constant set of IP addresses for your workload.

- Network Load Balancers can be used to distribute loads in a manner that integrates with AWS Auto Scaling to manage demand.

- Use a highly available DNS provider. DNS names allow your users to enter names instead of IP addresses to access your workloads and distributes this information to a defined scope, usually globally for users of the workload.

- Use Amazon Route 53 or a trusted DNS provider.

- [What is Amazon Route 53?](#)

- Use Route 53 to manage your CloudFront distributions and load balancers.

- Determine the domains and subdomains you are going to manage.

- Create appropriate record sets using ALIAS or CNAME records.

- Use the AWS global network to optimize the path from your users to your applications. AWS Global Accelerator continually monitors the health of your application endpoints and redirects traffic to healthy endpoints in less than 30 seconds.
 - AWS Global Accelerator is a service that improves the availability and performance of your applications with local or global users. It provides static IP addresses that act as a fixed entry point to your application endpoints in a single or multiple AWS Regions, such as your Application Load Balancers, Network Load Balancers or Amazon EC2 instances.
 - [What Is AWS Global Accelerator?](#)
- Configure and use Amazon CloudFront or a trusted content delivery network (CDN). A content delivery network can provide faster end-user response times and can serve requests for content that may cause unnecessary scaling of your workloads.
 - [What is Amazon CloudFront?](#)
 - Configure Amazon CloudFront distributions for your workloads, or use a third-party CDN.
 - You can limit access to your workloads so that they are only accessible from CloudFront by using the IP ranges for CloudFront in your endpoint security groups or access policies.

Resources

Related documents:

- [APN Partner: partners that can help you create automated compute solutions](#)
- [AWS Auto Scaling: How Scaling Plans Work](#)
- [AWS Marketplace: products that can be used with auto scaling](#)
- [Managing Throughput Capacity Automatically with DynamoDB Auto Scaling](#)
- [Using a load balancer with an Auto Scaling group](#)
- [What Is AWS Global Accelerator?](#)
- [What Is Amazon EC2 Auto Scaling?](#)
- [What is AWS Auto Scaling?](#)
- [What is Amazon CloudFront?](#)
- [What is Amazon Route 53?](#)
- [What is Elastic Load Balancing?](#)
- [What is a Network Load Balancer?](#)
- [What is an Application Load Balancer?](#)

- [Working with records](#)

REL07-BP02 Obtain resources upon detection of impairment to a workload

This best practice was updated with new guidance on December 6, 2023.

Scale resources reactively when necessary if availability is impacted, to restore workload availability.

You first must configure health checks and the criteria on these checks to indicate when availability is impacted by lack of resources. Then, either notify the appropriate personnel to manually scale the resource, or start automation to automatically scale it.

Scale can be manually adjusted for your workload (for example, changing the number of EC2 instances in an Auto Scaling group, or modifying throughput of a DynamoDB table through the AWS Management Console or AWS CLI). However, automation should be used whenever possible (refer to [**Use automation when obtaining or scaling resources**](#)).

Desired outcome: Scaling activities (either automatically or manually) are initiated to restore availability upon detection of a failure or degraded customer experience.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Implement observability and monitoring across all components in your workload, to monitor customer experience and detect failure. Define the procedures, manual or automated, that scale the required resources. For more information, see [REL11-BP01 Monitor all components of the workload to detect failures](#).

Implementation steps

- Define the procedures, manual or automated, that scale the required resources.
 - Scaling procedures depend on how the different components within your workload are designed.
 - Scaling procedures also vary depending on the underlying technology utilized.
 - Components using AWS Auto Scaling can use scaling plans to configure a set of instructions for scaling your resources. If you work with AWS CloudFormation or add tags to AWS

resources, you can set up scaling plans for different sets of resources per application. Auto Scaling provides recommendations for scaling strategies customized to each resource. After you create your scaling plan, Auto Scaling combines dynamic scaling and predictive scaling methods together to support your scaling strategy. For more detail, see [How scaling plans work](#).

- Amazon EC2 Auto Scaling verifies that you have the correct number of Amazon EC2 instances available to handle the load for your application. You create collections of EC2 instances, called Auto Scaling groups. You can specify the minimum and maximum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes below or above these limits. For more detail, see [What is Amazon EC2 Auto Scaling?](#)
- Amazon DynamoDB auto scaling uses the Application Auto Scaling service to dynamically adjust provisioned throughput capacity on your behalf, in response to actual traffic patterns. This allows a table or a global secondary index to increase its provisioned read and write capacity to handle sudden increases in traffic, without throttling. For more detail, see [Managing throughput capacity automatically with DynamoDB auto scaling](#).

Resources

Related best practices:

- [REL07-BP01 Use automation when obtaining or scaling resources](#)
- [REL11-BP01 Monitor all components of the workload to detect failures](#)

Related documents:

- [AWS Auto Scaling: How Scaling Plans Work](#)
- [Managing Throughput Capacity Automatically with DynamoDB Auto Scaling](#)
- [What Is Amazon EC2 Auto Scaling?](#)

REL07-BP03 Obtain resources upon detection that more resources are needed for a workload

Scale resources proactively to meet demand and avoid availability impact.

Many AWS services automatically scale to meet demand. If using Amazon EC2 instances or Amazon ECS clusters, you can configure automatic scaling of these to occur based on usage metrics that

correspond to demand for your workload. For Amazon EC2, average CPU utilization, load balancer request count, or network bandwidth can be used to scale out (or scale in) EC2 instances. For Amazon ECS, average CPU utilization, load balancer request count, and memory utilization can be used to scale out (or scale in) ECS tasks. Using Target Auto Scaling on AWS, the autoscaler acts like a household thermostat, adding or removing resources to maintain the target value (for example, 70% CPU utilization) that you specify.

Amazon EC2 Auto Scaling can also do [Predictive Auto Scaling](#), which uses machine learning to analyze each resource's historical workload and regularly forecasts the future load.

Little's Law helps calculate how many instances of compute (EC2 instances, concurrent Lambda functions, etc.) that you need.

$$L = \lambda W$$

L = number of instances (or mean concurrency in the system)

λ = mean rate at which requests arrive (req/sec)

W = mean time that each request spends in the system (sec)

For example, at 100 rps, if each request takes 0.5 seconds to process, you will need 50 instances to keep up with demand.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Obtain resources upon detection that more resources are needed for a workload. Scale resources proactively to meet demand and avoid availability impact.
 - Calculate how many compute resources you will need (compute concurrency) to handle a given request rate.
 - [Telling Stories About Little's Law](#)
 - When you have a historical pattern for usage, set up scheduled scaling for Amazon EC2 auto scaling.
 - [Scheduled Scaling for Amazon EC2 Auto Scaling](#)
 - Use AWS predictive scaling.
 - [Predictive scaling for Amazon EC2 Auto Scaling](#)

Resources

Related documents:

- [AWS Marketplace: products that can be used with auto scaling](#)
- [Managing Throughput Capacity Automatically with DynamoDB Auto Scaling](#)
- [Predictive Scaling for EC2, Powered by Machine Learning](#)
- [Scheduled Scaling for Amazon EC2 Auto Scaling](#)
- [Telling Stories About Little's Law](#)
- [What Is Amazon EC2 Auto Scaling?](#)

REL07-BP04 Load test your workload

Adopt a load testing methodology to measure if scaling activity meets workload requirements.

It's important to perform sustained load testing. Load tests should discover the breaking point and test the performance of your workload. AWS makes it easy to set up temporary testing environments that model the scale of your production workload. In the cloud, you can create a production-scale test environment on demand, complete your testing, and then decommission the resources. Because you only pay for the test environment when it's running, you can simulate your live environment for a fraction of the cost of testing on premises.

Load testing in production should also be considered as part of game days where the production system is stressed, during hours of lower customer usage, with all personnel on hand to interpret results and address any problems that arise.

Common anti-patterns:

- Performing load testing on deployments that are not the same configuration as your production.
- Performing load testing only on individual pieces of your workload, and not on the entire workload.
- Performing load testing with a subset of requests and not a representative set of actual requests.
- Performing load testing to a small safety factor above expected load.

Benefits of establishing this best practice: You know what components in your architecture fail under load and be able to identify what metrics to watch to indicate that you are approaching that load in time to address the problem, preventing the impact of that failure.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Perform load testing to identify which aspect of your workload indicates that you must add or remove capacity. Load testing should have representative traffic similar to what you receive in production. Increase the load while watching the metrics you have instrumented to determine which metric indicates when you must add or remove resources.
 - [Distributed Load Testing on AWS: simulate thousands of connected users](#)
 - Identify the mix of requests. You may have varied mixes of requests, so you should look at various time frames when identifying the mix of traffic.
 - Implement a load driver. You can use custom code, open source, or commercial software to implement a load driver.
 - Load test initially using small capacity. You see some immediate effects by driving load onto a lesser capacity, possibly as small as one instance or container.
 - Load test against larger capacity. The effects will be different on a distributed load, so you must test against as close to a product environment as possible.

Resources

Related documents:

- [Distributed Load Testing on AWS: simulate thousands of connected users](#)

REL 8. How do you implement change?

Controlled changes are necessary to deploy new functionality, and to verify that the workloads and the operating environment are running known software and can be patched or replaced in a predictable manner. If these changes are uncontrolled, then it makes it difficult to predict the effect of these changes, or to address issues that arise because of them.

Best practices

- [REL08-BP01 Use runbooks for standard activities such as deployment](#)
- [REL08-BP02 Integrate functional testing as part of your deployment](#)
- [REL08-BP03 Integrate resiliency testing as part of your deployment](#)
- [REL08-BP04 Deploy using immutable infrastructure](#)

- [REL08-BP05 Deploy changes with automation](#)

REL08-BP01 Use runbooks for standard activities such as deployment

Runbooks are the predefined procedures to achieve specific outcomes. Use runbooks to perform standard activities, whether done manually or automatically. Examples include deploying a workload, patching a workload, or making DNS modifications.

For example, put processes in place to [ensure rollback safety during deployments](#). Ensuring that you can roll back a deployment without any disruption for your customers is critical in making a service reliable.

For runbook procedures, start with a valid effective manual process, implement it in code, and invoke it to automatically run where appropriate.

Even for sophisticated workloads that are highly automated, runbooks are still useful for [running game days](#) or meeting rigorous reporting and auditing requirements.

Note that playbooks are used in response to specific incidents, and runbooks are used to achieve specific outcomes. Often, runbooks are for routine activities, while playbooks are used for responding to non-routine events.

Common anti-patterns:

- Performing unplanned changes to configuration in production.
- Skipping steps in your plan to deploy faster, resulting in a failed deployment.
- Making changes without testing the reversal of the change.

Benefits of establishing this best practice: Effective change planning increases your ability to successfully run the change because you are aware of all the systems impacted. Validating your change in test environments increases your confidence.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Provide consistent and prompt responses to well-understood events by documenting procedures in runbooks.
 - [AWS Well-Architected Framework: Concepts: Runbook](#)

- Use the principle of infrastructure as code to define your infrastructure. By using AWS CloudFormation (or a trusted third party) to define your infrastructure, you can use version control software to version and track changes.
 - Use AWS CloudFormation (or a trusted third-party provider) to define your infrastructure.
 - [What is AWS CloudFormation?](#)
 - Create templates that are singular and decoupled, using good software design principles.
 - Determine the permissions, templates, and responsible parties for implementation.
 - [Controlling access with AWS Identity and Access Management](#)
 - Use source control, like AWS CodeCommit or a trusted third-party tool, for version control.
 - [What is AWS CodeCommit?](#)

Resources

Related documents:

- [APN Partner: partners that can help you create automated deployment solutions](#)
- [AWS Marketplace: products that can be used to automate your deployments](#)
- [AWS Well-Architected Framework: Concepts: Runbook](#)
- [What is AWS CloudFormation?](#)
- [What is AWS CodeCommit?](#)

Related examples:

- [Automating operations with Playbooks and Runbooks](#)

REL08-BP02 Integrate functional testing as part of your deployment

Functional tests are run as part of automated deployment. If success criteria are not met, the pipeline is halted or rolled back.

These tests are run in a pre-production environment, which is staged prior to production in the pipeline. Ideally, this is done as part of a deployment pipeline.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Integrate functional testing as part of your deployment. Functional tests are run as part of automated deployment. If success criteria are not met, the pipeline is halted or rolled back.
 - Invoke AWS CodeBuild during the 'Test Action' of your software release pipelines modeled in AWS CodePipeline. This capability allows you to easily run a variety of tests against your code, such as unit tests, static code analysis, and integration tests.
 - [AWS CodePipeline Adds Support for Unit and Custom Integration Testing with AWS CodeBuild](#)
 - Use AWS Marketplace solutions for invoking automated tests as part of your software delivery pipeline.
 - [Software test automation](#)

Resources

Related documents:

- [AWS CodePipeline Adds Support for Unit and Custom Integration Testing with AWS CodeBuild](#)
- [Software test automation](#)
- [What Is AWS CodePipeline?](#)

REL08-BP03 Integrate resiliency testing as part of your deployment

Resiliency tests (using the [principles of chaos engineering](#)) are run as part of the automated deployment pipeline in a pre-production environment.

These tests are staged and run in the pipeline in a pre-production environment. They should also be run in production as part of [game days](#).

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Integrate resiliency testing as part of your deployment. Use Chaos Engineering, the discipline of experimenting on a workload to build confidence in the workload's capability to withstand turbulent conditions in production.
 - Resiliency tests inject faults or resource degradation to assess that your workload responds with its designed resilience.

- [Well-Architected lab: Level 300: Testing for Resiliency of EC2 RDS and S3](#)
- These tests can be run regularly in pre-production environments in automated deployment pipelines.
- They should also be run in production, as part of scheduled game days.
- Using Chaos Engineering principles, propose hypotheses about how your workload will perform under various impairments, then test your hypotheses using resiliency testing.
- [Principles of Chaos Engineering](#)

Resources

Related documents:

- [Principles of Chaos Engineering](#)
- [What is AWS Fault Injection Simulator?](#)

Related examples:

- [Well-Architected lab: Level 300: Testing for Resiliency of EC2 RDS and S3](#)

REL08-BP04 Deploy using immutable infrastructure

This best practice was updated with new guidance on December 6, 2023.

Immutable infrastructure is a model that mandates that no updates, security patches, or configuration changes happen in-place on production workloads. When a change is needed, the architecture is built onto new infrastructure and deployed into production.

Follow an immutable infrastructure deployment strategy to increase the reliability, consistency, and reproducibility in your workload deployments.

Desired outcome: With immutable infrastructure, no [in-place modifications](#) are allowed to run infrastructure resources within a workload. Instead, when a change is needed, a new set of updated infrastructure resources containing all the necessary changes are deployed in parallel to your existing resources. This deployment is validated automatically, and if successful, traffic is gradually shifted to the new set of resources.

This deployment strategy applies to software updates, security patches, infrastructure changes, configuration updates, and application updates, among others.

Common anti-patterns:

- Implementing in-place changes to running infrastructure resources.

Benefits of establishing this best practice:

- **Increased consistency across environments:** Since there are no differences in infrastructure resources across environments, consistency is increased and testing is simplified.
- **Reduction in configuration drifts:** By replacing infrastructure resources with a known and version-controlled configuration, the infrastructure is set to a known, tested, and trusted state, avoiding configuration drifts.
- **Reliable atomic deployments:** Deployments either complete successfully or nothing changes, increasing consistency and reliability in the deployment process.
- **Simplified deployments:** Deployments are simplified because they don't need to support upgrades. Upgrades are just new deployments.
- **Safer deployments with fast rollback and recovery processes:** Deployments are safer because the previous working version is not changed. You can roll back to it if errors are detected.
- **Enhanced security posture:** By not allowing changes to infrastructure, remote access mechanisms (such as SSH) can be disabled. This reduces the attack vector, improving your organization's security posture.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Automation

When defining an immutable infrastructure deployment strategy, it is recommended to use [automation](#) as much as possible to increase reproducibility and minimize the potential of human error. For more detail, see [REL08-BP05 Deploy changes with automation](#) and [Automating safe, hands-off deployments](#).

With [Infrastructure as code \(IaC\)](#), infrastructure provisioning, orchestration, and deployment steps are defined in a programmatic, descriptive, and declarative way and stored in a source control

system. Leveraging infrastructure as code makes it simpler to automate infrastructure deployment and helps achieve infrastructure immutability.

Deployment patterns

When a change in the workload is required, the immutable infrastructure deployment strategy mandates that a new set of infrastructure resources is deployed, including all necessary changes. It is important for this new set of resources to follow a rollout pattern that minimizes user impact. There are two main strategies for this deployment:

Canary deployment: The practice of directing a small number of your customers to the new version, usually running on a single service instance (the canary). You then deeply scrutinize any behavior changes or errors that are generated. You can remove traffic from the canary if you encounter critical problems and send the users back to the previous version. If the deployment is successful, you can continue to deploy at your desired velocity, while monitoring the changes for errors, until you are fully deployed. AWS CodeDeploy can be configured with a [deployment configuration](#) that allows a canary deployment.

Blue/green deployment: Similar to the canary deployment, except that a full fleet of the application is deployed in parallel. You alternate your deployments across the two stacks (blue and green). Once again, you can send traffic to the new version, and fall back to the old version if you see problems with the deployment. Commonly all traffic is switched at once, however you can also use fractions of your traffic to each version to dial up the adoption of the new version using the weighted DNS routing capabilities of Amazon Route 53. AWS CodeDeploy and [AWS Elastic Beanstalk](#) can be configured with a deployment configuration that allows a blue/green deployment.

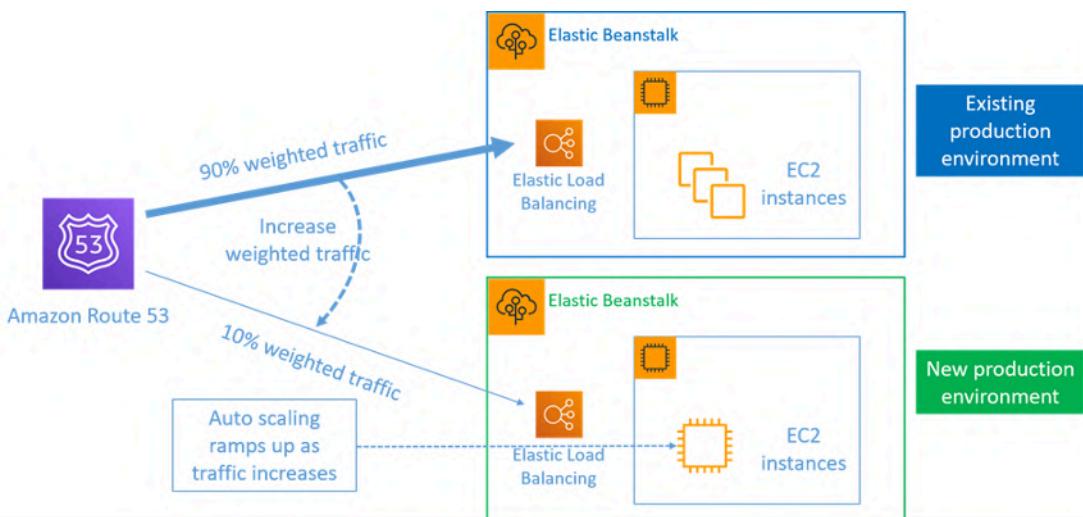


Figure 8: Blue/green deployment with AWS Elastic Beanstalk and Amazon Route 53

Drift detection

Drift is defined as any change that causes an infrastructure resource to have a different state or configuration to what is expected. Any type of unmanaged configuration change goes against the notion of immutable infrastructure, and should be detected and remediated in order to have a successful implementation of immutable infrastructure.

Implementation steps

- Disallow the in-place modification of running infrastructure resources.
 - You can use [AWS Identity and Access Management \(IAM\)](#) to specify who or what can access services and resources in AWS, centrally manage fine-grained permissions, and analyze access to refine permissions across AWS.
- Automate the deployment of infrastructure resources to increase reproducibility and minimize the potential of human error.
 - As described in the [Introduction to DevOps on AWS whitepaper](#), automation is a cornerstone with AWS services and is internally supported in all services, features, and offerings.
 - [*Prebaking*](#) your Amazon Machine Image (AMI) can speed up the time to launch them. [EC2 Image Builder](#) is a fully managed AWS service that helps you automate the creation, maintenance, validation, sharing, and deployment of customized, secure, and up-to-date Linux or Windows custom AMI.
 - Some of the services that support automation are:
 - [AWS Elastic Beanstalk](#) is a service to rapidly deploy and scale web applications developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, NGINX, Passenger, and IIS.
 - [AWS Proton](#) helps platform teams connect and coordinate all the different tools your development teams need for infrastructure provisioning, code deployments, monitoring, and updates. AWS Proton enables automated infrastructure as code provisioning and deployment of serverless and container-based applications.
 - Leveraging infrastructure as code makes it easy to automate infrastructure deployment, and helps achieve infrastructure immutability. AWS provides services that enable the creation, deployment, and maintenance of infrastructure in a programmatic, descriptive, and declarative way.
 - [AWS CloudFormation](#) helps developers create AWS resources in an orderly and predictable fashion. Resources are written in text files using JSON or YAML format. The templates require a specific syntax and structure that depends on the types of resources being created

and managed. You author your resources in JSON or YAML with any code editor such as AWS Cloud9, check it into a version control system, and then CloudFormation builds the specified services in safe, repeatable manner.

- [AWS Serverless Application Model \(AWS SAM\)](#) is an open-source framework that you can use to build serverless applications on AWS. AWS SAM integrates with other AWS services, and is an extension of AWS CloudFormation.
- [AWS Cloud Development Kit \(AWS CDK\)](#) is an open-source software development framework to model and provision your cloud application resources using familiar programming languages. You can use AWS CDK to model application infrastructure using TypeScript, Python, Java, and .NET. AWS CDK uses AWS CloudFormation in the background to provision resources in a safe, repeatable manner.
- [AWS Cloud Control API](#) introduces a common set of Create, Read, Update, Delete, and List (CRUDL) APIs to help developers manage their cloud infrastructure in an easy and consistent way. The Cloud Control API common APIs allow developers to uniformly manage the lifecycle of AWS and third-party services.
- Implement deployment patterns that minimize user impact.
 - Canary deployments:
 - [Set up an API Gateway canary release deployment](#)
 - [Create a pipeline with canary deployments for Amazon ECS using AWS App Mesh](#)
 - Blue/green deployments: the [Blue/Green Deployments on AWS whitepaper](#) describes [example techniques](#) to implement blue/green deployment strategies.
 - Detect configuration or state drifts. For more detail, see [Detecting unmanaged configuration changes to stacks and resources](#).

Resources

Related best practices:

- [REL08-BP05 Deploy changes with automation](#)

Related documents:

- [Automating safe, hands-off deployments](#)
- [Leveraging AWS CloudFormation to create an immutable infrastructure at Nubank](#)
- [Infrastructure as code](#)

- [Implementing an alarm to automatically detect drift in AWS CloudFormation stacks](#)

Related videos:

- [AWS re:Invent 2020: Reliability, consistency, and confidence through immutability](#)

REL08-BP05 Deploy changes with automation

Deployments and patching are automated to eliminate negative impact.

Making changes to production systems is one of the largest risk areas for many organizations. We consider deployments a first-class problem to be solved alongside the business problems that the software addresses. Today, this means the use of automation wherever practical in operations, including testing and deploying changes, adding or removing capacity, and migrating data.

AWS CodePipeline lets you manage the steps required to release your workload. This includes a deployment state using AWS CodeDeploy to automate deployment of application code to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.

Recommendation

Although conventional wisdom suggests that you keep humans in the loop for the most difficult operational procedures, we suggest that you automate the most difficult procedures for that very reason.

Common anti-patterns:

- Manually performing changes.
- Skipping steps in your automation through emergency work flows.
- Not following your plans.

Benefits of establishing this best practice: Using automation to deploy all changes removes the potential for introduction of human error and provides the ability to test before changing production to ensure that your plans are complete.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Automate your deployment pipeline. Deployment pipelines allow you to invoke automated testing and detection of anomalies, and either halt the pipeline at a certain step before production deployment, or automatically roll back a change.
 - [The Amazon Builders' Library: Ensuring rollback safety during deployments](#)
 - [The Amazon Builders' Library: Going faster with continuous delivery](#)
 - Use AWS CodePipeline (or a trusted third-party product) to define and run your pipelines.
 - Configure the pipeline to start when a change is committed to your code repository.
 - [What is AWS CodePipeline?](#)
 - Use Amazon Simple Notification Service (Amazon SNS) and Amazon Simple Email Service (Amazon SES) to send notifications about problems in the pipeline or integrate with a team chat tool, like Amazon Chime.
 - [What is Amazon Simple Notification Service?](#)
 - [What is Amazon SES?](#)
 - [What is Amazon Chime?](#)
 - [Automate chat messages with webhooks.](#)

Resources

Related documents:

- [APN Partner: partners that can help you create automated deployment solutions](#)
- [AWS Marketplace: products that can be used to automate your deployments](#)
- [Automate chat messages with webhooks.](#)
- [The Amazon Builders' Library: Ensuring rollback safety during deployments](#)
- [The Amazon Builders' Library: Going faster with continuous delivery](#)
- [What Is AWS CodePipeline?](#)
- [What Is CodeDeploy?](#)
- [AWS Systems Manager Patch Manager](#)
- [What is Amazon SES?](#)
- [What is Amazon Simple Notification Service?](#)

Related videos:

- [AWS Summit 2019: CI/CD on AWS](#)

Failure management

Questions

- [REL 9. How do you back up data?](#)
- [REL 10. How do you use fault isolation to protect your workload?](#)
- [REL 11. How do you design your workload to withstand component failures?](#)
- [REL 12. How do you test reliability?](#)
- [REL 13. How do you plan for disaster recovery \(DR\)?](#)

REL 9. How do you back up data?

Back up data, applications, and configuration to meet your requirements for recovery time objectives (RTO) and recovery point objectives (RPO).

Best practices

- [REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources](#)
- [REL09-BP02 Secure and encrypt backups](#)
- [REL09-BP03 Perform data backup automatically](#)
- [REL09-BP04 Perform periodic recovery of the data to verify backup integrity and processes](#)

REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources

Understand and use the backup capabilities of the data services and resources used by the workload. Most services provide capabilities to back up workload data.

Desired outcome: Data sources have been identified and classified based on criticality. Then, establish a strategy for data recovery based on the RPO. This strategy involves either backing up these data sources, or having the ability to reproduce data from other sources. In the case of data loss, the strategy implemented allows recovery or the reproduction of data within the defined RPO and RTO.

Cloud maturity phase: Foundational

Common anti-patterns:

- Not aware of all data sources for the workload and their criticality.
- Not taking backups of critical data sources.
- Taking backups of only some data sources without using criticality as a criterion.
- No defined RPO, or backup frequency cannot meet RPO.
- Not evaluating if a backup is necessary or if data can be reproduced from other sources.

Benefits of establishing this best practice: Identifying the places where backups are necessary and implementing a mechanism to create backups, or being able to reproduce the data from an external source improves the ability to restore and recover data during an outage.

Level of risk exposed if this best practice is not established: High

Implementation guidance

All AWS data stores offer backup capabilities. Services such as Amazon RDS and Amazon DynamoDB additionally support automated backup that allows point-in-time recovery (PITR), which allows you to restore a backup to any time up to five minutes or less before the current time. Many AWS services offer the ability to copy backups to another AWS Region. AWS Backup is a tool that gives you the ability to centralize and automate data protection across AWS services.

[AWS Elastic Disaster Recovery](#) allows you to copy full server workloads and maintain continuous data protection from on-premise, cross-AZ or cross-Region, with a Recovery Point Objective (RPO) measured in seconds.

Amazon S3 can be used as a backup destination for self-managed and AWS-managed data sources. AWS services such as Amazon EBS, Amazon RDS, and Amazon DynamoDB have built in capabilities to create backups. Third-party backup software can also be used.

On-premises data can be backed up to the AWS Cloud using [AWS Storage Gateway](#) or [AWS DataSync](#). Amazon S3 buckets can be used to store this data on AWS. Amazon S3 offers multiple storage tiers such as [Amazon S3 Glacier](#) or [S3 Glacier Deep Archive](#) to reduce cost of data storage.

You might be able to meet data recovery needs by reproducing the data from other sources. For example, [Amazon ElastiCache replica nodes](#) or [Amazon RDS read replicas](#) could be used to reproduce data if the primary is lost. In cases where sources like this can be used to meet your [Recovery Point Objective \(RPO\)](#) and [Recovery Time Objective \(RTO\)](#), you might not require a

backup. Another example, if working with Amazon EMR, it might not be necessary to backup your HDFS data store, as long as you can [reproduce the data into Amazon EMR from Amazon S3](#).

When selecting a backup strategy, consider the time it takes to recover data. The time needed to recover data depends on the type of backup (in the case of a backup strategy), or the complexity of the data reproduction mechanism. This time should fall within the RTO for the workload.

Implementation steps

- 1. Identify all data sources for the workload.** Data can be stored on a number of resources such as [databases](#), [volumes](#), [filesystems](#), [logging systems](#), and [object storage](#). Refer to the **Resources** section to find **Related documents** on different AWS services where data is stored, and the backup capability these services provide.
- 2. Classify data sources based on criticality.** Different data sets will have different levels of criticality for a workload, and therefore different requirements for resiliency. For example, some data might be critical and require a RPO near zero, while other data might be less critical and can tolerate a higher RPO and some data loss. Similarly, different data sets might have different RTO requirements as well.
- 3. Use AWS or third-party services to create backups of the data.** [AWS Backup](#) is a managed service that allows creating backups of various data sources on AWS. [AWS Elastic Disaster Recovery](#) handles automated sub-second data replication to an AWS Region. Most AWS services also have native capabilities to create backups. The AWS Marketplace has many solutions that provide these capabilities as well. Refer to the **Resources** listed below for information on how to create backups of data from various AWS services.
- 4. For data that is not backed up, establish a data reproduction mechanism.** You might choose not to backup data that can be reproduced from other sources for various reasons. There might be a situation where it is cheaper to reproduce data from sources when needed rather than creating a backup as there may be a cost associated with storing backups. Another example is where restoring from a backup takes longer than reproducing the data from sources, resulting in a breach in RTO. In such situations, consider tradeoffs and establish a well-defined process for how data can be reproduced from these sources when data recovery is necessary. For example, if you have loaded data from Amazon S3 to a data warehouse (like Amazon Redshift), or MapReduce cluster (like Amazon EMR) to do analysis on that data, this may be an example of data that can be reproduced from other sources. As long as the results of these analyses are either stored somewhere or reproducible, you would not suffer a data loss from a failure in the data warehouse or MapReduce cluster. Other examples that can be reproduced from sources include caches (like Amazon ElastiCache) or RDS read replicas.

5. Establish a cadence for backing up data. Creating backups of data sources is a periodic process and the frequency should depend on the RPO.

Level of effort for the Implementation Plan: Moderate

Resources

Related Best Practices:

[REL13-BP01 Define recovery objectives for downtime and data loss](#)

[REL13-BP02 Use defined recovery strategies to meet the recovery objectives](#)

Related documents:

- [What Is AWS Backup?](#)
- [What is AWS DataSync?](#)
- [What is Volume Gateway?](#)
- [APN Partner: partners that can help with backup](#)
- [AWS Marketplace: products that can be used for backup](#)
- [Amazon EBS Snapshots](#)
- [Backing Up Amazon EFS](#)
- [Backing up Amazon FSx for Windows File Server](#)
- [Backup and Restore for ElastiCache for Redis](#)
- [Creating a DB Cluster Snapshot in Neptune](#)
- [Creating a DB Snapshot](#)
- [Creating an EventBridge Rule That Triggers on a Schedule](#)
- [Cross-Region Replication with Amazon S3](#)
- [EFS-to-EFS AWS Backup](#)
- [Exporting Log Data to Amazon S3](#)
- [Object lifecycle management](#)
- [On-Demand Backup and Restore for DynamoDB](#)
- [Point-in-time recovery for DynamoDB](#)
- [Working with Amazon OpenSearch Service Index Snapshots](#)
- [What is AWS Elastic Disaster Recovery?](#)

Related videos:

- [AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS](#)
- [AWS Backup Demo: Cross-Account and Cross-Region Backup](#)
- [AWS re:Invent 2019: Deep dive on AWS Backup, ft. Rackspace \(STG341\)](#)

Related examples:

- [Well-Architected Lab - Implementing Bi-Directional Cross-Region Replication \(CRR\) for Amazon S3](#)
- [Well-Architected Lab - Testing Backup and Restore of Data](#)
- [Well-Architected Lab - Backup and Restore with Fallback for Analytics Workload](#)
- [Well-Architected Lab - Disaster Recovery - Backup and Restore](#)

REL09-BP02 Secure and encrypt backups

Control and detect access to backups using authentication and authorization. Prevent and detect if data integrity of backups is compromised using encryption.

Common anti-patterns:

- Having the same access to the backups and restoration automation as you do to the data.
- Not encrypting your backups.

Benefits of establishing this best practice: Securing your backups prevents tampering with the data, and encryption of the data prevents access to that data if it is accidentally exposed.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Control and detect access to backups using authentication and authorization, such as AWS Identity and Access Management (IAM). Prevent and detect if data integrity of backups is compromised using encryption.

Amazon S3 supports several methods of encryption of your data at rest. Using server-side encryption, Amazon S3 accepts your objects as unencrypted data, and then encrypts them as they are stored. Using client-side encryption, your workload application is responsible for encrypting the

data before it is sent to Amazon S3. Both methods allow you to use AWS Key Management Service (AWS KMS) to create and store the data key, or you can provide your own key, which you are then responsible for. Using AWS KMS, you can set policies using IAM on who can and cannot access your data keys and decrypted data.

For Amazon RDS, if you have chosen to encrypt your databases, then your backups are encrypted also. DynamoDB backups are always encrypted. When using AWS Elastic Disaster Recovery, all data in transit and at rest is encrypted. With Elastic Disaster Recovery, data at rest can be encrypted using either the default Amazon EBS encryption Volume Encryption Key or a custom customer-managed key.

Implementation steps

1. Use encryption on each of your data stores. If your source data is encrypted, then the backup will also be encrypted.
 - [Use encryption in Amazon RDS..](#) You can configure encryption at rest using AWS Key Management Service when you create an RDS instance.
 - [Use encryption on Amazon EBS volumes..](#) You can configure default encryption or specify a unique key upon volume creation.
 - Use the required [Amazon DynamoDB encryption](#). DynamoDB encrypts all data at rest. You can either use an AWS owned AWS KMS key or an AWS managed KMS key, specifying a key that is stored in your account.
 - [Encrypt your data stored in Amazon EFS](#). Configure the encryption when you create your file system.
 - Configure the encryption in the source and destination Regions. You can configure encryption at rest in Amazon S3 using keys stored in KMS, but the keys are Region-specific. You can specify the destination keys when you configure the replication.
 - Choose whether to use the default or custom [Amazon EBS encryption for Elastic Disaster Recovery](#). This option will encrypt your replicated data at rest on the Staging Area Subnet disks and the replicated disks.
2. Implement least privilege permissions to access your backups. Follow best practices to limit the access to the backups, snapshots, and replicas in accordance with [security best practices](#).

Resources

Related documents:

- [AWS Marketplace: products that can be used for backup](#)
- [Amazon EBS Encryption](#)
- [Amazon S3: Protecting Data Using Encryption](#)
- [CRR Additional Configuration: Replicating Objects Created with Server-Side Encryption \(SSE\) Using Encryption Keys stored in AWS KMS](#)
- [DynamoDB Encryption at Rest](#)
- [Encrypting Amazon RDS Resources](#)
- [Encrypting Data and Metadata in Amazon EFS](#)
- [Encryption for Backups in AWS](#)
- [Managing Encrypted Tables](#)
- [Security Pillar - AWS Well-Architected Framework](#)
- [What is AWS Elastic Disaster Recovery?](#)

Related examples:

- [Well-Architected Lab - Implementing Bi-Directional Cross-Region Replication \(CRR\) for Amazon S3](#)

REL09-BP03 Perform data backup automatically

Configure backups to be taken automatically based on a periodic schedule informed by the Recovery Point Objective (RPO), or by changes in the dataset. Critical datasets with low data loss requirements need to be backed up automatically on a frequent basis, whereas less critical data where some loss is acceptable can be backed up less frequently.

Desired outcome: An automated process that creates backups of data sources at an established cadence.

Common anti-patterns:

- Performing backups manually.
- Using resources that have backup capability, but not including the backup in your automation.

Benefits of establishing this best practice: Automating backups verifies that they are taken regularly based on your RPO, and alerts you if they are not taken.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

AWS Backup can be used to create automated data backups of various AWS data sources. Amazon RDS instances can be backed up almost continuously every five minutes and Amazon S3 objects can be backed up almost continuously every fifteen minutes, providing for point-in-time recovery (PITR) to a specific point in time within the backup history. For other AWS data sources, such as Amazon EBS volumes, Amazon DynamoDB tables, or Amazon FSx file systems, AWS Backup can run automated backup as frequently as every hour. These services also offer native backup capabilities. AWS services that offer automated backup with point-in-time recovery include [Amazon DynamoDB](#), [Amazon RDS](#), and [Amazon Keyspaces \(for Apache Cassandra\)](#) – these can be restored to a specific point in time within the backup history. Most other AWS data storage services offer the ability to schedule periodic backups, as frequently as every hour.

Amazon RDS and Amazon DynamoDB offer continuous backup with point-in-time recovery. Amazon S3 versioning, once turned on, is automatic. [Amazon Data Lifecycle Manager](#) can be used to automate the creation, copy and deletion of Amazon EBS snapshots. It can also automate the creation, copy, deprecation and deregistration of Amazon EBS-backed Amazon Machine Images (AMIs) and their underlying Amazon EBS snapshots.

AWS Elastic Disaster Recovery provides continuous block-level replication from the source environment (on-premises or AWS) to the target recovery region. Point-in-time Amazon EBS snapshots are automatically created and managed by the service.

For a centralized view of your backup automation and history, AWS Backup provides a fully managed, policy-based backup solution. It centralizes and automates the back up of data across multiple AWS services in the cloud as well as on premises using the AWS Storage Gateway.

In addition to versioning, Amazon S3 features replication. The entire S3 bucket can be automatically replicated to another bucket in the same, or a different AWS Region.

Implementation steps

- 1. Identify data sources** that are currently being backed up manually. For more detail, see [REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources](#).
- 2. Determine the RPO** for the workload. For more detail, see [REL13-BP01 Define recovery objectives for downtime and data loss](#).

3. **Use an automated backup solution or managed service.** AWS Backup is a fully-managed service that makes it easy to [centralize and automate data protection across AWS services, in the cloud, and on-premises](#). Using backup plans in AWS Backup, create rules which define the resources to backup, and the frequency at which these backups should be created. This frequency should be informed by the RPO established in Step 2. For hands-on guidance on how to create automated backups using AWS Backup, see [Testing Backup and Restore of Data](#). Native backup capabilities are offered by most AWS services that store data. For example, RDS can be leveraged for automated backups with point-in-time recovery (PITR).
4. **For data sources not supported** by an automated backup solution or managed service such as on-premises data sources or message queues, consider using a trusted third-party solution to create automated backups. Alternatively, you can create automation to do this using the AWS CLI or SDKs. You can use AWS Lambda Functions or AWS Step Functions to define the logic involved in creating a data backup, and use Amazon EventBridge to invoke it at a frequency based on your RPO.

Level of effort for the Implementation Plan: Low

Resources

Related documents:

- [APN Partner: partners that can help with backup](#)
- [AWS Marketplace: products that can be used for backup](#)
- [Creating an EventBridge Rule That Triggers on a Schedule](#)
- [What Is AWS Backup?](#)
- [What Is AWS Step Functions?](#)
- [What is AWS Elastic Disaster Recovery?](#)

Related videos:

- [AWS re:Invent 2019: Deep dive on AWS Backup, ft. Rackspace \(STG341\)](#)

Related examples:

- [Well-Architected Lab - Testing Backup and Restore of Data](#)

REL09-BP04 Perform periodic recovery of the data to verify backup integrity and processes

Validate that your backup process implementation meets your Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO) by performing a recovery test.

Desired outcome: Data from backups is periodically recovered using well-defined mechanisms to verify that recovery is possible within the established recovery time objective (RTO) for the workload. Verify that restoration from a backup results in a resource that contains the original data without any of it being corrupted or inaccessible, and with data loss within the recovery point objective (RPO).

Common anti-patterns:

- Restoring a backup, but not querying or retrieving any data to check that the restoration is usable.
- Assuming that a backup exists.
- Assuming that the backup of a system is fully operational and that data can be recovered from it.
- Assuming that the time to restore or recover data from a backup falls within the RTO for the workload.
- Assuming that the data contained on the backup falls within the RPO for the workload
- Restoring when necessary, without using a runbook or outside of an established automated procedure.

Benefits of establishing this best practice: Testing the recovery of the backups verifies that data can be restored when needed without having any worry that data might be missing or corrupted, that the restoration and recovery is possible within the RTO for the workload, and any data loss falls within the RPO for the workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Testing backup and restore capability increases confidence in the ability to perform these actions during an outage. Periodically restore backups to a new location and run tests to verify the integrity of the data. Some common tests that should be performed are checking if all data is available, is not corrupted, is accessible, and that any data loss falls within the RPO for the workload. Such tests can also help ascertain if recovery mechanisms are fast enough to accommodate the workload's RTO.

Using AWS, you can stand up a testing environment and restore your backups to assess RTO and RPO capabilities, and run tests on data content and integrity.

Additionally, Amazon RDS and Amazon DynamoDB allow point-in-time recovery (PITR). Using continuous backup, you can restore your dataset to the state it was in at a specified date and time.

If all the data is available, is not corrupted, is accessible, and any data loss falls within the RPO for the workload. Such tests can also help ascertain if recovery mechanisms are fast enough to accommodate the workload's RTO.

AWS Elastic Disaster Recovery offers continual point-in-time recovery snapshots of Amazon EBS volumes. As source servers are replicated, point-in-time states are chronicled over time based on the configured policy. Elastic Disaster Recovery helps you verify the integrity of these snapshots by launching instances for test and drill purposes without redirecting the traffic.

Implementation steps

- 1. Identify data sources** that are currently being backed up and where these backups are being stored. For implementation guidance, see [REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources](#).
- 2. Establish criteria for data validation** for each data source. Different types of data will have different properties which might require different validation mechanisms. Consider how this data might be validated before you are confident to use it in production. Some common ways to validate data are using data and backup properties such as data type, format, checksum, size, or a combination of these with custom validation logic. For example, this might be a comparison of the checksum values between the restored resource and the data source at the time the backup was created.
- 3. Establish RTO and RPO** for restoring the data based on data criticality. For implementation guidance, see [REL13-BP01 Define recovery objectives for downtime and data loss](#).
- 4. Assess your recovery capability.** Review your backup and restore strategy to understand if it can meet your RTO and RPO, and adjust the strategy as necessary. Using [AWS Resilience Hub](#), you can run an assessment of your workload. The assessment evaluates your application configuration against the resiliency policy and reports if your RTO and RPO targets can be met.
- 5. Do a test restore** using currently established processes used in production for data restoration. These processes depend on how the original data source was backed up, the format and storage location of the backup itself, or if the data is reproduced from other sources. For example, if you are using a managed service such as [AWS Backup, this might be as simple as restoring the](#)

[backup into a new resource](#). If you used AWS Elastic Disaster Recovery you can [launch a recovery drill](#).

6. **Validate data recovery** from the restored resource based on criteria you previously established for data validation. Does the restored and recovered data contain the most recent record or item at the time of backup? Does this data fall within the RPO for the workload?
7. **Measure time required** for restore and recovery and compare it to your established RTO. Does this process fall within the RTO for the workload? For example, compare the timestamps from when the restoration process started and when the recovery validation completed to calculate how long this process takes. All AWS API calls are timestamped and this information is available in [AWS CloudTrail](#). While this information can provide details on when the restore process started, the end timestamp for when the validation was completed should be recorded by your validation logic. If using an automated process, then services like [Amazon DynamoDB](#) can be used to store this information. Additionally, many AWS services provide an event history which provides timestamped information when certain actions occurred. Within AWS Backup, backup and restore actions are referred to as *jobs*, and these jobs contain timestamp information as part of its metadata which can be used to measure time required for restoration and recovery.
8. **Notify stakeholders** if data validation fails, or if the time required for restoration and recovery exceeds the established RTO for the workload. When implementing automation to do this, [such as in this lab](#), services like Amazon Simple Notification Service (Amazon SNS) can be used to send push notifications such as email or SMS to stakeholders. [These messages can also be published to messaging applications such as Amazon Chime, Slack, or Microsoft Teams](#) or used to [create tasks as OpsItems using AWS Systems Manager OpsCenter](#).
9. **Automate this process to run periodically**. For example, services like AWS Lambda or a State Machine in AWS Step Functions can be used to automate the restore and recovery processes, and Amazon EventBridge can be used to invoke this automation workflow periodically as shown in the architecture diagram below. Learn how to [Automate data recovery validation with AWS Backup](#). Additionally, [this Well-Architected lab](#) provides a hands-on experience on one way to do automation for several of the steps here.

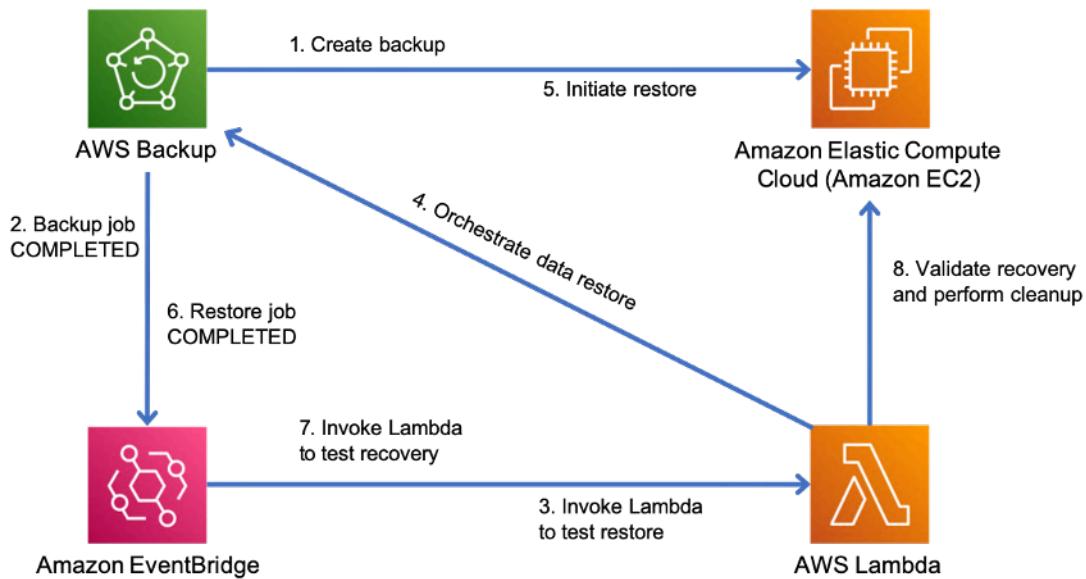


Figure 9. An automated backup and restore process

Level of effort for the Implementation Plan: Moderate to high depending on the complexity of the validation criteria.

Resources

Related documents:

- [Automate data recovery validation with AWS Backup](#)
- [APN Partner: partners that can help with backup](#)
- [AWS Marketplace: products that can be used for backup](#)
- [Creating an EventBridge Rule That Triggers on a Schedule](#)
- [On-demand backup and restore for DynamoDB](#)
- [What Is AWS Backup?](#)
- [What Is AWS Step Functions?](#)
- [What is AWS Elastic Disaster Recovery](#)
- [AWS Elastic Disaster Recovery](#)

Related examples:

- [Well-Architected lab: Testing Backup and Restore of Data](#)

REL 10. How do you use fault isolation to protect your workload?

Fault isolated boundaries limit the effect of a failure within a workload to a limited number of components. Components outside of the boundary are unaffected by the failure. Using multiple fault isolated boundaries, you can limit the impact on your workload.

Best practices

- [REL10-BP01 Deploy the workload to multiple locations](#)
- [REL10-BP02 Select the appropriate locations for your multi-location deployment](#)
- [REL10-BP03 Automate recovery for components constrained to a single location](#)
- [REL10-BP04 Use bulkhead architectures to limit scope of impact](#)

REL10-BP01 Deploy the workload to multiple locations

Distribute workload data and resources across multiple Availability Zones or, where necessary, across AWS Regions. These locations can be as diverse as required.

One of the bedrock principles for service design in AWS is the avoidance of single points of failure in underlying physical infrastructure. This motivates us to build software and systems that use multiple Availability Zones and are resilient to failure of a single zone. Similarly, systems are built to be resilient to failure of a single compute node, single storage volume, or single instance of a database. When building a system that relies on redundant components, it's important to ensure that the components operate independently, and in the case of AWS Regions, autonomously. The benefits achieved from theoretical availability calculations with redundant components are only valid if this holds true.

Availability Zones (AZs)

AWS Regions are composed of multiple Availability Zones that are designed to be independent of each other. Each Availability Zone is separated by a meaningful physical distance from other zones to avoid correlated failure scenarios due to environmental hazards like fires, floods, and tornadoes. Each Availability Zone also has independent physical infrastructure: dedicated connections to utility power, standalone backup power sources, independent mechanical services, and independent network connectivity within and beyond the Availability Zone. This design limits faults in any of these systems to just the one affected AZ. Despite being geographically separated, Availability Zones are located in the same regional area which allows high-throughput, low-latency networking. The entire AWS Region (across all Availability Zones, consisting of multiple physically

independent data centers) can be treated as a single logical deployment target for your workload, including the ability to synchronously replicate data (for example, between databases). This allows you to use Availability Zones in an active/active or active/standby configuration.

Availability Zones are independent, and therefore workload availability is increased when the workload is architected to use multiple zones. Some AWS services (including the Amazon EC2 instance data plane) are deployed as strictly zonal services where they have shared fate with the Availability Zone they are in. Amazon EC2 instances in the other AZs will however be unaffected and continue to function. Similarly, if a failure in an Availability Zone causes an Amazon Aurora database to fail, a read-replica Aurora instance in an unaffected AZ can be automatically promoted to primary. Regional AWS services, such as Amazon DynamoDB on the other hand internally use multiple Availability Zones in an active/active configuration to achieve the availability design goals for that service, without you needing to configure AZ placement.

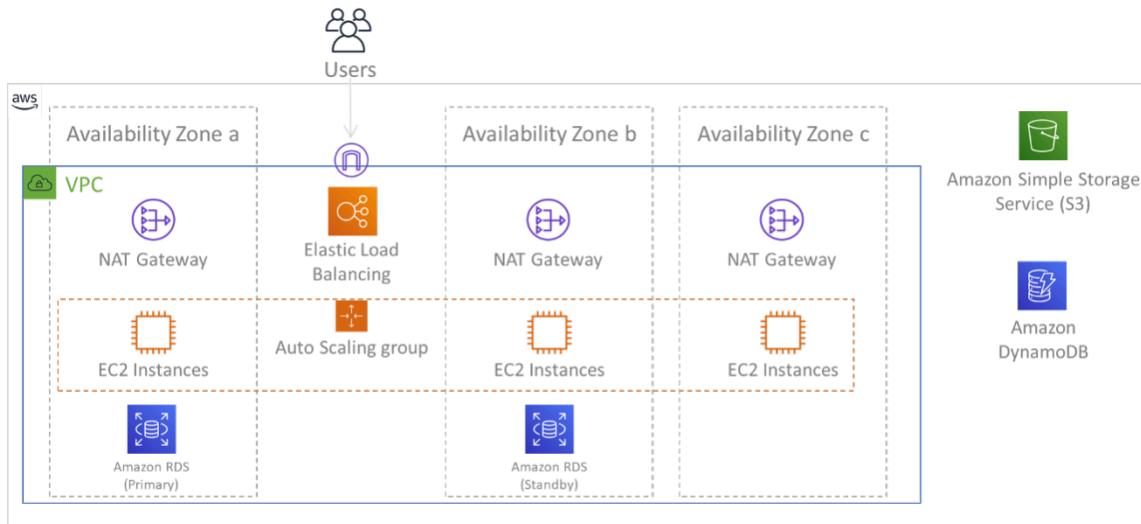


Figure 9: Multi-tier architecture deployed across three Availability Zones. Note that Amazon S3 and Amazon DynamoDB are always Multi-AZ automatically. The ELB also is deployed to all three zones.

While AWS control planes typically provide the ability to manage resources within the entire Region (multiple Availability Zones), certain control planes (including Amazon EC2 and Amazon EBS) have the ability to filter results to a single Availability Zone. When this is done, the request is processed only in the specified Availability Zone, reducing exposure to disruption in other Availability Zones. This AWS CLI example illustrates getting Amazon EC2 instance information from only the us-east-2c Availability Zone:

```
AWS ec2 describe-instances --filters Name=availability-zone,Values=us-east-2c
```

AWS Local Zones

AWS Local Zones act similarly to Availability Zones within their respective AWS Region in that they can be selected as a placement location for zonal AWS resources such as subnets and EC2 instances. What makes them special is that they are located not in the associated AWS Region, but near large population, industry, and IT centers where no AWS Region exists today. Yet they still retain high-bandwidth, secure connection between local workloads in the local zone and those running in the AWS Region. You should use AWS Local Zones to deploy workloads closer to your users for low-latency requirements.

Amazon Global Edge Network

Amazon Global Edge Network consists of edge locations in cities around the world. Amazon CloudFront uses this network to deliver content to end users with lower latency. AWS Global Accelerator allows you to create your workload endpoints in these edge locations to provide onboarding to the AWS global network close to your users. Amazon API Gateway allows edge-optimized API endpoints using a CloudFront distribution to facilitate client access through the closest edge location.

AWS Regions

AWS Regions are designed to be autonomous, therefore, to use a multi-Region approach you would deploy dedicated copies of services to each Region.

A multi-Region approach is common for *disaster recovery* strategies to meet recovery objectives when one-off large-scale events occur. See [Plan for Disaster Recovery \(DR\)](#) for more information on these strategies. Here however, we focus instead on *availability*, which seeks to deliver a mean uptime objective over time. For high-availability objectives, a multi-region architecture will generally be designed to be active/active, where each service copy (in their respective regions) is active (serving requests).

Recommendation

Availability goals for most workloads can be satisfied using a Multi-AZ strategy within a single AWS Region. Consider multi-Region architectures only when workloads have extreme availability requirements, or other business goals, that require a multi-Region architecture.

AWS provides you with the capabilities to operate services cross-region. For example, AWS provides continuous, asynchronous data replication of data using Amazon Simple Storage Service

(Amazon S3) Replication, Amazon RDS Read Replicas (including Aurora Read Replicas), and Amazon DynamoDB Global Tables. With continuous replication, versions of your data are available for near immediate use in each of your active Regions.

Using AWS CloudFormation, you can define your infrastructure and deploy it consistently across AWS accounts and across AWS Regions. And AWS CloudFormation StackSets extends this functionality by allowing you to create, update, or delete AWS CloudFormation stacks across multiple accounts and regions with a single operation. For Amazon EC2 instance deployments, an AMI (Amazon Machine Image) is used to supply information such as hardware configuration and installed software. You can implement an Amazon EC2 Image Builder pipeline that creates the AMIs you need and copy these to your active regions. This ensures that these *Golden AMIs* have everything you need to deploy and scale-out your workload in each new region.

To route traffic, both Amazon Route 53 and AWS Global Accelerator permit the definition of policies that determine which users go to which active regional endpoint. With Global Accelerator you set a traffic dial to control the percentage of traffic that is directed to each application endpoint. Route 53 supports this percentage approach, and also multiple other available policies including geoproximity and latency based ones. Global Accelerator automatically leverages the extensive network of AWS edge servers, to onboard traffic to the AWS network backbone as soon as possible, resulting in lower request latencies.

All of these capabilities operate so as to preserve each Region's autonomy. There are very few exceptions to this approach, including our services that provide global edge delivery (such as Amazon CloudFront and Amazon Route 53), along with the control plane for the AWS Identity and Access Management (IAM) service. Most services operate entirely within a single Region.

On-premises data center

For workloads that run in an on-premises data center, architect a hybrid experience when possible. AWS Direct Connect provides a dedicated network connection from your premises to AWS allowing you to run in both.

Another option is to run AWS infrastructure and services on premises using AWS Outposts. AWS Outposts is a fully managed service that extends AWS infrastructure, AWS services, APIs, and tools to your data center. The same hardware infrastructure used in the AWS Cloud is installed in your data center. AWS Outposts are then connected to the nearest AWS Region. You can then use AWS Outposts to support your workloads that have low latency or local data processing requirements.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Use multiple Availability Zones and AWS Regions. Distribute workload data and resources across multiple Availability Zones or, where necessary, across AWS Regions. These locations can be as diverse as required.
 - Regional services are inherently deployed across Availability Zones.
 - This includes Amazon S3, Amazon DynamoDB, and AWS Lambda (when not connected to a VPC)
 - Deploy your container, instance, and function-based workloads into multiple Availability Zones. Use multi-zone datastores, including caches. Use the features of Amazon EC2 Auto Scaling, Amazon ECS task placement, AWS Lambda function configuration when running in your VPC, and ElastiCache clusters.
 - Use subnets that are in separate Availability Zones when you deploy Auto Scaling groups.
 - [Example: Distributing instances across Availability Zones](#)
 - [Choosing Regions and Availability Zones](#)
 - Use ECS task placement parameters, specifying DB subnet groups.
 - [Amazon ECS task placement strategies](#)
 - Use subnets in multiple Availability Zones when you configure a function to run in your VPC.
 - [Configuring an AWS Lambda function to access resources in an Amazon VPC](#)
 - Use multiple Availability Zones with ElastiCache clusters.
 - [Choosing Regions and Availability Zones](#)
 - If your workload must be deployed to multiple Regions, choose a multi-Region strategy. Most reliability needs can be met within a single AWS Region using a multi-Availability Zone strategy. Use a multi-Region strategy when necessary to meet your business needs.
 - [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications \(ARC209-R2\)](#)
 - Backup to another AWS Region can add another layer of assurance that data will be available when needed.
 - Some workloads have regulatory requirements that require use of a multi-Region strategy.
 - Evaluate AWS Outposts for your workload. If your workload requires low latency to your on-premises data center or has local data processing requirements. Then run AWS infrastructure and services on premises using AWS Outposts
 - [What is AWS Outposts?](#)

- Determine if AWS Local Zones helps you provide service to your users. If you have low-latency requirements, see if AWS Local Zones is located near your users. If yes, then use it to deploy workloads closer to those users.
 - [AWS Local Zones FAQ](#)

Resources

Related documents:

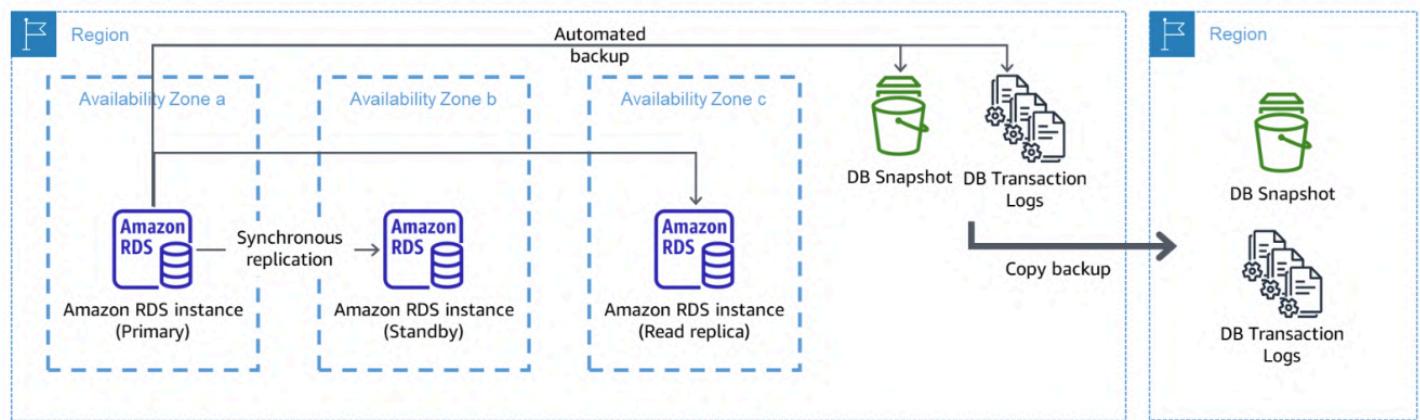
- [AWS Global Infrastructure](#)
- [AWS Local Zones FAQ](#)
- [Amazon ECS task placement strategies](#)
- [Choosing Regions and Availability Zones](#)
- [Example: Distributing instances across Availability Zones](#)
- [Global Tables: Multi-Region Replication with DynamoDB](#)
- [Using Amazon Aurora global databases](#)
- [Creating a Multi-Region Application with AWS Services blog series](#)
- [What is AWS Outposts?](#)

Related videos:

- [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications \(ARC209-R2\)](#)
- [AWS re:Invent 2019: Innovation and operation of the AWS global network infrastructure \(NET339\)](#)

REL10-BP02 Select the appropriate locations for your multi-location deployment

Desired outcome: For high availability, always (when possible) deploy your workload components to multiple Availability Zones (AZs). For workloads with extreme resilience requirements, carefully evaluate the options for a multi-Region architecture.



A resilient multi-AZ database deployment with backup to another AWS Region

Common anti-patterns:

- Choosing to design a multi-Region architecture when a multi-AZ architecture would satisfy requirements.
- Not accounting for dependencies between application components if resilience and multi-location requirements differ between those components.

benefits of establishing this best practice: For resilience, you should use an approach that builds layers of defense. One layer protects against smaller, more common, disruptions by building a highly available architecture using multiple AZs. Another layer of defense is meant to protect against rare events like widespread natural disasters and Region-level disruptions. This second layer involves architecting your application to span multiple AWS Regions.

- The difference between a 99.5% availability and 99.99% availability is over 3.5 hours per month. The expected availability of a workload can only reach “four nines” if it is in multiple AZs.
- By running your workload in multiple AZs, you can isolate faults in power, cooling, and networking, and most natural disasters like fire and flood.
- Implementing a multi-Region strategy for your workload helps protect it against widespread natural disasters that affect a large geographic region of a country, or technical failures of Region-wide scope. Be aware that implementing a multi-Region architecture can be significantly complex, and is usually not required for most workloads.

Level of risk exposed if this best practice is not established: High

Implementation guidance

For a disaster event based on disruption or partial loss of one Availability Zone, implementing a highly available workload in multiple Availability Zones within a single AWS Region helps mitigate against natural and technical disasters. Each AWS Region is comprised of multiple Availability Zones, each isolated from faults in the other zones and separated by a meaningful distance. However, for a disaster event that includes the risk of losing multiple Availability Zone components, which are a significant distance away from each other, you should implement disaster recovery options to mitigate against failures of a Region-wide scope. For workloads that require extreme resilience (critical infrastructure, health-related applications, financial system infrastructure, etc.), a multi-Region strategy may be required.

Implementation Steps

1. Evaluate your workload and determine whether the resilience needs can be met by a multi-AZ approach (single AWS Region), or if they require a multi-Region approach. Implementing a multi-Region architecture to satisfy these requirements will introduce additional complexity, therefore carefully consider your use case and its requirements. Resilience requirements can almost always be met using a single AWS Region. Consider the following possible requirements when determining whether you need to use multiple Regions:
 - a. **Disaster recovery (DR):** For a disaster event based on disruption or partial loss of one Availability Zone, implementing a highly available workload in multiple Availability Zones within a single AWS Region helps mitigate against natural and technical disasters. For a disaster event that includes the risk of losing multiple Availability Zone components, which are a significant distance away from each other, you should implement disaster recovery across multiple Regions to mitigate against natural disasters or technical failures of a Region-wide scope.
 - b. **High availability (HA):** A multi-Region architecture (using multiple AZs in each Region) can be used to achieve greater than four 9's (> 99.99%) availability.
 - c. **Stack localization:** When deploying a workload to a global audience, you can deploy localized stacks in different AWS Regions to serve audiences in those Regions. Localization can include language, currency, and types of data stored.
 - d. **Proximity to users:** When deploying a workload to a global audience, you can reduce latency by deploying stacks in AWS Regions close to where the end users are.
 - e. **Data residency:** Some workloads are subject to data residency requirements, where data from certain users must remain within a specific country's borders. Based on the regulation in

question, you can choose to deploy an entire stack, or just the data, to the AWS Region within those borders.

2. Here are some examples of multi-AZ functionality provided by AWS services:

- a. To protect workloads using EC2 or ECS, deploy an Elastic Load Balancer in front of the compute resources. Elastic Load Balancing then provides the solution to detect instances in unhealthy zones and route traffic to the healthy ones.
 - i. [Getting started with Application Load Balancers](#)
 - ii. [Getting started with Network Load Balancers](#)
- b. In the case of EC2 instances running commercial off-the-shelf software that do not support load balancing, you can achieve a form of fault tolerance by implementing a multi-AZ disaster recovery methodology.
 - i. [the section called “REL13-BP02 Use defined recovery strategies to meet the recovery objectives”](#)
- c. For Amazon ECS tasks, deploy your service evenly across three AZs to achieve a balance of availability and cost.
 - i. [Amazon ECS availability best practices | Containers](#)
- d. For non-Aurora Amazon RDS, you can choose Multi-AZ as a configuration option. Upon failure of the primary database instance, Amazon RDS automatically promotes a standby database to receive traffic in another availability zone. Multi-Region read-replicas can also be created to improve resilience.
 - i. [Amazon RDS Multi AZ Deployments](#)
 - ii. [Creating a read replica in a different AWS Region](#)

3. Here are some examples of multi-Region functionality provided by AWS services:

- a. For Amazon S3 workloads, where multi-AZ availability is provided automatically by the service, consider Multi-Region Access Points if a multi-Region deployment is needed.
 - i. [Multi-Region Access Points in Amazon S3](#)
- b. For DynamoDB tables, where multi-AZ availability is provided automatically by the service, you can easily convert existing tables to global tables to take advantage of multiple regions.
 - i. [Convert Your Single-Region Amazon DynamoDB Tables to Global Tables](#)
- c. If your workload is fronted by Application Load Balancers or Network Load Balancers, use AWS Global Accelerator to improve the availability of your application by directing traffic to multiple regions that contain healthy endpoints.

- i. [Endpoints for standard accelerators in AWS Global Accelerator - AWS Global Accelerator \(amazon.com\)](#)
- d. For applications that leverage AWS EventBridge, consider cross-Region buses to forward events to other Regions you select.
 - i. [Sending and receiving Amazon EventBridge events between AWS Regions](#)
- e. For Amazon Aurora databases, consider Aurora global databases, which span multiple AWS regions. Existing clusters can be modified to add new Regions as well.
 - i. [Getting started with Amazon Aurora global databases](#)
- f. If your workload includes AWS Key Management Service (AWS KMS) encryption keys, consider whether multi-Region keys are appropriate for your application.
 - i. [Multi-Region keys in AWS KMS](#)
- g. For other AWS service features, see this blog series on [Creating a Multi-Region Application with AWS Services series](#)

Level of effort for the Implementation Plan: Moderate to High

Resources

Related documents:

- [Creating a Multi-Region Application with AWS Services series](#)
- [Disaster Recovery \(DR\) Architecture on AWS, Part IV: Multi-site Active/Active](#)
- [AWS Global Infrastructure](#)
- [AWS Local Zones FAQ](#)
- [Disaster Recovery \(DR\) Architecture on AWS, Part I: Strategies for Recovery in the Cloud](#)
- [Disaster recovery is different in the cloud](#)
- [Global Tables: Multi-Region Replication with DynamoDB](#)

Related videos:

- [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications \(ARC209-R2\)](#)
- [Auth0: Multi-Region High-Availability Architecture that Scales to 1.5B+ Logins a Month with automated failover](#)

Related examples:

- [Disaster Recovery \(DR\) Architecture on AWS, Part I: Strategies for Recovery in the Cloud](#)
- [DTCC achieves resilience well beyond what they can do on premises](#)
- [Expedia Group uses a multi-Region, multi-Availability Zone architecture with a proprietary DNS service to add resilience to the applications](#)
- [Uber: Disaster Recovery for Multi-Region Kafka](#)
- [Netflix: Active-Active for Multi-Regional Resilience](#)
- [How we build Data Residency for Atlassian Cloud](#)
- [Intuit TurboTax runs across two Regions](#)

REL10-BP03 Automate recovery for components constrained to a single location

If components of the workload can only run in a single Availability Zone or in an on-premises data center, implement the capability to do a complete rebuild of the workload within your defined recovery objectives.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

If the best practice to deploy the workload to multiple locations is not possible due to technological constraints, you must implement an alternate path to resiliency. You must automate the ability to recreate necessary infrastructure, redeploy applications, and recreate necessary data for these cases.

For example, Amazon EMR launches all nodes for a given cluster in the same Availability Zone because running a cluster in the same zone improves performance of the jobs flows as it provides a higher data access rate. If this component is required for workload resilience, then you must have a way to redeploy the cluster and its data. Also for Amazon EMR, you should provision redundancy in ways other than using Multi-AZ. You can provision [multiple nodes](#). Using [EMR File System \(EMRFS\)](#), data in EMR can be stored in Amazon S3, which in turn can be replicated across multiple Availability Zones or AWS Regions.

Similarly, for Amazon Redshift, by default it provisions your cluster in a randomly selected Availability Zone within the AWS Region that you select. All the cluster nodes are provisioned in the same zone.

For stateful server-based workloads deployed to an on-premise data center, you can use AWS Elastic Disaster Recovery to protect your workloads in AWS. If you are already hosted in AWS, you can use Elastic Disaster Recovery to protect your workload to an alternative Availability Zone or Region. Elastic Disaster Recovery uses continual block-level replication to a lightweight staging area to provide fast, reliable recovery of on-premises and cloud-based applications.

Implementation steps

1. Implement self-healing. Deploy your instances or containers using automatic scaling when possible. If you cannot use automatic scaling, use automatic recovery for EC2 instances or implement self-healing automation based on Amazon EC2 or ECS container lifecycle events.
 - Use [Amazon EC2 Auto Scaling groups](#) for instances and container workloads that have no requirements for a single instance IP address, private IP address, Elastic IP address, and instance metadata.
 - The launch template user data can be used to implement automation that can self-heal most workloads.
 - Use automatic [recovery of Amazon EC2 instances](#) for workloads that require a single instance ID address, private IP address, elastic IP address, and instance metadata.
 - Automatic Recovery will send recovery status alerts to a SNS topic as the instance failure is detected.
 - Use [Amazon EC2 instance lifecycle events](#) or [Amazon ECS events](#) to automate self-healing where automatic scaling or EC2 recovery cannot be used.
 - Use the events to invoke automation that will heal your component according to the process logic you require.
 - Protect stateful workloads that are limited to a single location using [AWS Elastic Disaster Recovery](#).

Resources

Related documents:

- [Amazon ECS events](#)
- [Amazon EC2 Auto Scaling lifecycle hooks](#)
- [Recover your instance.](#)
- [Service automatic scaling](#)
- [What Is Amazon EC2 Auto Scaling?](#)

- [AWS Elastic Disaster Recovery](#)

REL10-BP04 Use bulkhead architectures to limit scope of impact

Implement bulkhead architectures (also known as cell-based architectures) to restrict the effect of failure within a workload to a limited number of components.

Desired outcome: A cell-based architecture uses multiple isolated instances of a workload, where each instance is known as a cell. Each cell is independent, does not share state with other cells, and handles a subset of the overall workload requests. This reduces the potential impact of a failure, such as a bad software update, to an individual cell and the requests it is processing. If a workload uses 10 cells to service 100 requests, when a failure occurs, 90% of the overall requests would be unaffected by the failure.

Common anti-patterns:

- Allowing cells to grow without bounds.
- Applying code updates or deployments to all cells at the same time.
- Sharing state or components between cells (with the exception of the router layer).
- Adding complex business or routing logic to the router layer.
- Not minimizing cross-cell interactions.

Benefits of establishing this best practice: With cell-based architectures, many common types of failure are contained within the cell itself, providing additional fault isolation. These fault boundaries can provide resilience against failure types that otherwise are hard to contain, such as unsuccessful code deployments or requests that are corrupted or invoke a specific failure mode (also known as *poison pill requests*).

Level of risk exposed if this best practice is not established: High

Implementation guidance

On a ship, bulkheads ensure that a hull breach is contained within one section of the hull. In complex systems, this pattern is often replicated to allow fault isolation. Fault isolated boundaries restrict the effect of a failure within a workload to a limited number of components. Components outside of the boundary are unaffected by the failure. Using multiple fault isolated boundaries, you can limit the impact on your workload. On AWS, customers can use multiple Availability Zones

and Regions to provide fault isolation, but the concept of fault isolation can be extended to your workload's architecture as well.

The overall workload is partitioned cells by a partition key. This key needs to align with the *grain* of the service, or the natural way that a service's workload can be subdivided with minimal cross-cell interactions. Examples of partition keys are customer ID, resource ID, or any other parameter easily accessible in most API calls. A cell routing layer distributes requests to individual cells based on the partition key and presents a single endpoint to clients.

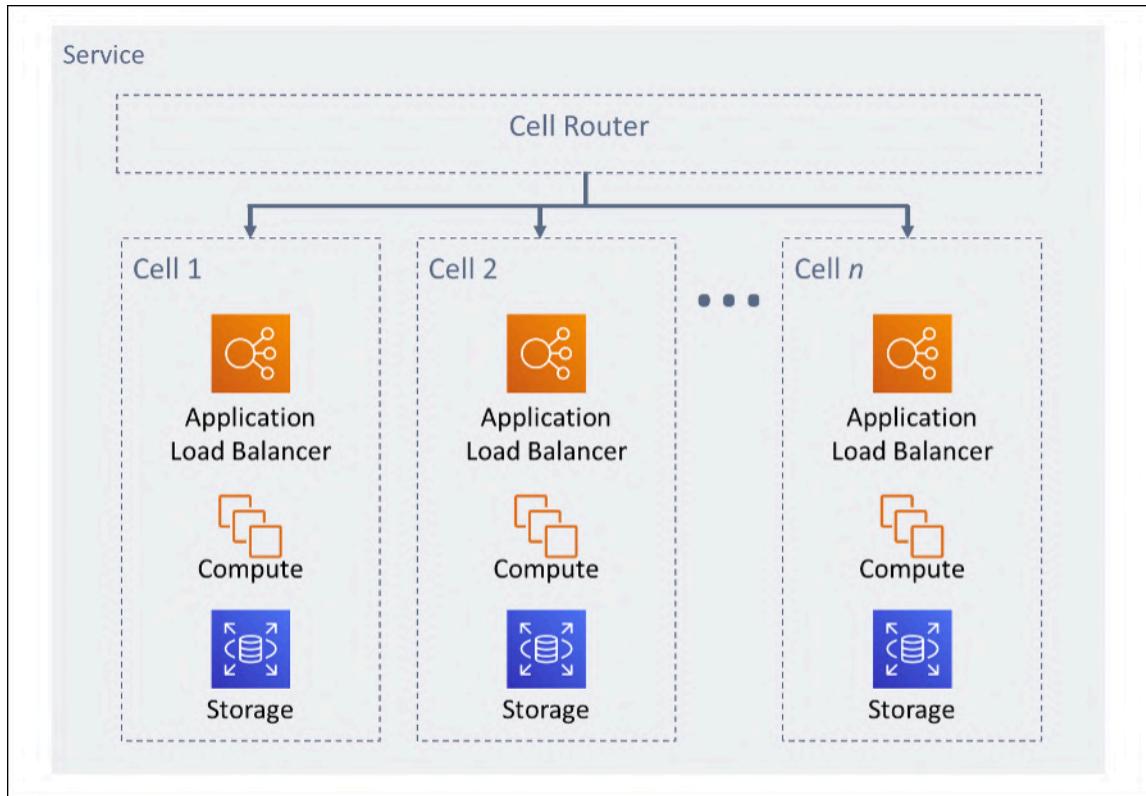


Figure 11: Cell-based architecture

Implementation steps

When designing a cell-based architecture, there are several design considerations to consider:

1. Partition key: Special consideration should be taken while choosing the partition key.

- It should align with the grain of the service, or the natural way that a service's workload can be subdivided with minimal cross-cell interactions. Examples are customer ID or resource ID.
- The partition key must be available in all requests, either directly or in a way that could be easily inferred deterministically by other parameters.

2. Persistent cell mapping: Upstream services should only interact with a single cell for the lifecycle of their resources.

- Depending on the workload, a cell migration strategy may be needed to migrate data from one cell to another. A possible scenario when a cell migration may be needed is if a particular user or resource in your workload becomes too big and requires it to have a dedicated cell.
- Cells should not share state or components between cells.
- Consequently, cross-cell interactions should be avoided or kept to a minimum, as those interactions create dependencies between cells and therefore diminish the fault isolation improvements.

3. Router layer: The router layer is a shared component between cells, and therefore cannot follow the same compartmentalization strategy as with cells.

- It is recommended for the router layer to distribute requests to individual cells using a partition mapping algorithm in a computationally efficient manner, such as combining cryptographic hash functions and modular arithmetic to map partition keys to cells.
- To avoid multi-cell impacts, the routing layer must remain as simple and horizontally scalable as possible, which necessitates avoiding complex business logic within this layer. This has the added benefit of making it easy to understand its expected behavior at all times, allowing for thorough testability. As explained by Colm MacCárthaigh in [Reliability, constant work, and a good cup of coffee](#), simple designs and constant work patterns produce reliable systems and reduce anti-fragility.

4. Cell size: Cells should have a maximum size and should not be allowed to grow beyond it.

- The maximum size should be identified by performing thorough testing, until breaking points are reached and safe operating margins are established. For more detail on how to implement testing practices, see [REL07-BP04 Load test your workload](#)
- The overall workload should grow by adding additional cells, allowing the workload to scale with increases in demand.

5. Multi-AZ or Multi-Region strategies: Multiple layers of resilience should be leveraged to protect against different failure domains.

- For resilience, you should use an approach that builds layers of defense. One layer protects against smaller, more common disruptions by building a highly available architecture using multiple AZs. Another layer of defense is meant to protect against rare events like widespread natural disasters and Region-level disruptions. This second layer involves architecting your application to span multiple AWS Regions. Implementing a multi-Region strategy for your workload helps protect it against widespread natural disasters that affect a large geographic

region of a country, or technical failures of Region-wide scope. Be aware that implementing a multi-Region architecture can be significantly complex, and is usually not required for most workloads. For more detail, see [REL10-BP02 Select the appropriate locations for your multi-location deployment](#).

- 6. Code deployment:** A staggered code deployment strategy should be preferred over deploying code changes to all cells at the same time.
- This helps minimize potential failure to multiple cells due to a bad deployment or human error. For more detail, see [Automating safe, hands-off deployment](#).

Resources

Related best practices:

- [REL07-BP04 Load test your workload](#)
- [REL10-BP02 Select the appropriate locations for your multi-location deployment](#)

Related documents:

- [Reliability, constant work, and a good cup of coffee](#)
- [AWS and Compartmentalization](#)
- [Workload isolation using shuffle-sharding](#)
- [Automating safe, hands-off deployment](#)

Related videos:

- [AWS re:Invent 2018: Close Loops and Opening Minds: How to Take Control of Systems, Big and Small](#)
- [AWS re:Invent 2018: How AWS Minimizes the Blast Radius of Failures \(ARC338\)](#)
- [Shuffle-sharding: AWS re:Invent 2019: Introducing The Amazon Builders' Library \(DOP328\)](#)
- [AWS Summit ANZ 2021 - Everything fails, all the time: Designing for resilience](#)

Related examples:

- [Well-Architected Lab - Fault isolation with shuffle sharding](#)

REL 11. How do you design your workload to withstand component failures?

Workloads with a requirement for high availability and low mean time to recovery (MTTR) must be architected for resiliency.

Best practices

- [REL11-BP01 Monitor all components of the workload to detect failures](#)
- [REL11-BP02 Fail over to healthy resources](#)
- [REL11-BP03 Automate healing on all layers](#)
- [REL11-BP04 Rely on the data plane and not the control plane during recovery](#)
- [REL11-BP05 Use static stability to prevent bimodal behavior](#)
- [REL11-BP06 Send notifications when events impact availability](#)
- [REL11-BP07 Architect your product to meet availability targets and uptime service level agreements \(SLAs\)](#)

REL11-BP01 Monitor all components of the workload to detect failures

Continually monitor the health of your workload so that you and your automated systems are aware of failures or degradations as soon as they occur. Monitor for key performance indicators (KPIs) based on business value.

All recovery and healing mechanisms must start with the ability to detect problems quickly. Technical failures should be detected first so that they can be resolved. However, availability is based on the ability of your workload to deliver business value, so key performance indicators (KPIs) that measure this need to be a part of your detection and remediation strategy.

Desired outcome: Essential components of a workload are monitored independently to detect and alert on failures when and where they happen.

Common anti-patterns:

- No alarms have been configured, so outages occur without notification.
- Alarms exist, but at thresholds that don't provide adequate time to react.
- Metrics are not collected often enough to meet the recovery time objective (RTO).
- Only the customer facing interfaces of the workload are actively monitored.
- Only collecting technical metrics, no business function metrics.

- No metrics measuring the user experience of the workload.
- Too many monitors are created.

Benefits of establishing this best practice: Having appropriate monitoring at all layers allows you to reduce recovery time by reducing time to detection.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Identify all workloads that will be reviewed for monitoring. Once you have identified all components of the workload that will need to be monitored, you will now need to determine the monitoring interval. The monitoring interval will have a direct impact on how fast recovery can be initiated based on the time it takes to detect a failure. The mean time to detection (MTTD) is the amount of time between a failure occurring and when repair operations begin. The list of services should be extensive and complete.

Monitoring must cover all layers of the application stack including application, platform, infrastructure, and network.

Your monitoring strategy should consider the impact of *gray failures*. For more detail on gray failures, see [Gray failures](#) in the Advanced Multi-AZ Resilience Patterns whitepaper.

Implementation steps

- Your monitoring interval is dependent on how quickly you must recover. Your recovery time is driven by the time it takes to recover, so you must determine the frequency of collection by accounting for this time and your recovery time objective (RTO).
- Configure detailed monitoring for components and managed services.
 - Determine if [detailed monitoring for EC2 instances](#) and [Auto Scaling](#) is necessary. Detailed monitoring provides one minute interval metrics, and default monitoring provides five minute interval metrics.
 - Determine if [enhanced monitoring](#) for RDS is necessary. Enhanced monitoring uses an agent on RDS instances to get useful information about different process or threads.
 - Determine the monitoring requirements of critical serverless components for [Lambda](#), [API Gateway](#), [Amazon EKS](#), [Amazon ECS](#), and all types of [load balancers](#).
 - Determine the monitoring requirements of storage components for [Amazon S3](#), [Amazon FSx](#), [Amazon EFS](#), and [Amazon EBS](#).

- Create [custom metrics](#) to measure business key performance indicators (KPIs). Workloads implement key business functions, which should be used as KPIs that help identify when an indirect problem happens.
- Monitor the user experience for failures using user canaries. [Synthetic transaction testing](#) (also known as canary testing, but not to be confused with canary deployments) that can run and simulate customer behavior is among the most important testing processes. Run these tests constantly against your workload endpoints from diverse remote locations.
- Create [custom metrics](#) that track the user's experience. If you can instrument the experience of the customer, you can determine when the consumer experience degrades.
- Set [alarms](#) to detect when any part of your workload is not working properly and to indicate when to automatically scale resources. Alarms can be visually displayed on dashboards, send alerts through Amazon SNS or email, and work with Auto Scaling to scale workload resources up or down.
- Create [dashboards](#) to visualize your metrics. Dashboards can be used to visually see trends, outliers, and other indicators of potential problems or to provide an indication of problems you may want to investigate.
- Create [distributed tracing monitoring](#) for your services. With distributed monitoring, you can understand how your application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors.
- Create monitoring systems (using [CloudWatch](#) or [X-Ray](#)) dashboards and data collection in a separate Region and account.
- Create integration for [Amazon Health Aware](#) monitoring to allow for monitoring visibility to AWS resources that might have degradations. For business essential workloads, this solution provides access to proactive and real-time alerts for AWS services.

Resources

Related best practices:

- [Availability Definition](#)
- [REL11-BP06 Send Notifications when events impact availability](#)

Related documents:

- [Amazon CloudWatch Synthetics enables you to create user canaries](#)

- [Enable or Disable Detailed Monitoring for Your Instance](#)
- [Enhanced Monitoring](#)
- [Monitoring Your Auto Scaling Groups and Instances Using Amazon CloudWatch](#)
- [Publishing Custom Metrics](#)
- [Using Amazon CloudWatch Alarms](#)
- [Using CloudWatch Dashboards](#)
- [Using Cross Region Cross Account CloudWatch Dashboards](#)
- [Using Cross Region Cross Account X-Ray Tracing](#)
- [Understanding availability](#)
- [Implementing Amazon Health Aware \(AHA\)](#)

Related videos:

- [Mitigating gray failures](#)

Related examples:

- [Well-Architected Lab: Level 300: Implementing Health Checks and Managing Dependencies to Improve Reliability](#)
- [One Observability Workshop: Explore X-Ray](#)

Related tools:

- [CloudWatch](#)
- [CloudWatch X-Ray](#)

REL11-BP02 Fail over to healthy resources

If a resource failure occurs, healthy resources should continue to serve requests. For location impairments (such as Availability Zone or AWS Region), ensure that you have systems in place to fail over to healthy resources in unimpaired locations.

When designing a service, distribute load across resources, Availability Zones, or Regions. Therefore, failure of an individual resource or impairment can be mitigated by shifting traffic to

remaining healthy resources. Consider how services are discovered and routed to in the event of a failure.

Design your services with fault recovery in mind. At AWS, we design services to minimize the time to recover from failures and impact on data. Our services primarily use data stores that acknowledge requests only after they are durably stored across multiple replicas within a Region. They are constructed to use cell-based isolation and use the fault isolation provided by Availability Zones. We use automation extensively in our operational procedures. We also optimize our replace-and-restart functionality to recover quickly from interruptions.

The patterns and designs that allow for the failover vary for each AWS platform service. Many AWS native managed services are natively multiple Availability Zone (like Lambda or API Gateway). Other AWS services (like EC2 and EKS) require specific best practice designs to support failover of resources or data storage across AZs.

Monitoring should be set up to check that the failover resource is healthy, track the progress of the resources failing over, and monitor business process recovery.

Desired outcome: Systems are capable of automatically or manually using new resources to recover from degradation.

Common anti-patterns:

- Planning for failure is not part of the planning and design phase.
- RTO and RPO are not established.
- Insufficient monitoring to detect failing resources.
- Proper isolation of failure domains.
- Multi-Region fail over is not considered.
- Detection for failure is too sensitive or aggressive when deciding to failover.
- Not testing or validating failover design.
- Performing auto healing automation, but not notifying that healing was needed.
- Lack of dampening period to avoid failing back too soon.

Benefits of establishing this best practice: You can build more resilient systems that maintain reliability when experiencing failures by degrading gracefully and recovering quickly.

Level of risk exposed if this best practice is not established: High

Implementation guidance

AWS services, such as [Elastic Load Balancing](#) and [Amazon EC2 Auto Scaling](#), help distribute load across resources and Availability Zones. Therefore, failure of an individual resource (such as an EC2 instance) or impairment of an Availability Zone can be mitigated by shifting traffic to remaining healthy resources.

For multi-Region workloads, designs are more complicated. For example, cross-Region read replicas allow you to deploy your data to multiple AWS Regions. However, failover is still required to promote the read replica to primary and then point your traffic to the new endpoint. Amazon Route 53, Route 53 Route 53 ARC, CloudFront, and AWS Global Accelerator can help route traffic across AWS Regions.

AWS services, such as Amazon S3, Lambda, API Gateway, Amazon SQS, Amazon SNS, Amazon SES, Amazon Pinpoint, Amazon ECR, AWS Certificate Manager, EventBridge, or Amazon DynamoDB, are automatically deployed to multiple Availability Zones by AWS. In case of failure, these AWS services automatically route traffic to healthy locations. Data is redundantly stored in multiple Availability Zones and remains available.

For Amazon RDS, Amazon Aurora, Amazon Redshift, Amazon EKS, or Amazon ECS, Multi-AZ is a configuration option. AWS can direct traffic to the healthy instance if failover is initiated. This failover action may be taken by AWS or as required by the customer

For Amazon EC2 instances, Amazon Redshift, Amazon ECS tasks, or Amazon EKS pods, you choose which Availability Zones to deploy to. For some designs, Elastic Load Balancing provides the solution to detect instances in unhealthy zones and route traffic to the healthy ones. Elastic Load Balancing can also route traffic to components in your on-premises data center.

For Multi-Region traffic failover, rerouting can leverage Amazon Route 53, Route 53 ARC, AWS Global Accelerator, Route 53 Private DNS for VPCs, or CloudFront to provide a way to define internet domains and assign routing policies, including health checks, to route traffic to healthy Regions. AWS Global Accelerator provides static IP addresses that act as a fixed entry point to your application, then route to endpoints in AWS Regions of your choosing, using the AWS global network instead of the internet for better performance and reliability.

Implementation steps

- Create failover designs for all appropriate applications and services. Isolate each architecture component and create failover designs meeting RTO and RPO for each component.

- Configure lower environments (like development or test) with all services that are required to have a failover plan. Deploy the solutions using infrastructure as code (IaC) to ensure repeatability.
- Configure a recovery site such as a second Region to implement and test the failover designs. If necessary, resources for testing can be configured temporarily to limit additional costs.
- Determine which failover plans are automated by AWS, which can be automated by a DevOps process, and which might be manual. Document and measure each service's RTO and RPO.
- Create a failover playbook and include all steps to failover each resource, application, and service.
- Create a fallback playbook and include all steps to fallback (with timing) each resource, application, and service
- Create a plan to initiate and rehearse the playbook. Use simulations and chaos testing to test the playbook steps and automation.
- For location impairment (such as Availability Zone or AWS Region), ensure you have systems in place to fail over to healthy resources in unimpaired locations. Check quota, autoscaling levels, and resources running before failover testing.

Resources

Related Well-Architected best practices:

- [REL13- Plan for DR](#)
- [REL10 - Use fault isolation to protect your workload](#)

Related documents:

- [Setting RTO and RPO Targets](#)
- [Set up Route 53 ARC with application loadbalancers](#)
- [Failover using Route 53 Weighted routing](#)
- [DR with Route 53 ARC](#)
- [EC2 with autoscaling](#)
- [EC2 Deployments - Multi-AZ](#)
- [ECS Deployments - Multi-AZ](#)
- [Switch traffic using Route 53 ARC](#)

- [Lambda with an Application Load Balancer and Failover](#)
- [ACM Replication and Failover](#)
- [Parameter Store Replication and Failover](#)
- [ECR cross region replication and Failover](#)
- [Secrets manager cross region replication configuration](#)
- [Enable cross region replication for EFS and Failover](#)
- [EFS Cross Region Replication and Failover](#)
- [Networking Failover](#)
- [S3 Endpoint failover using MRAP](#)
- [Create cross region replication for S3](#)
- [Failover Regional API Gateway with Route 53 ARC](#)
- [Failover using multi-region global accelerator](#)
- [Failover with DRS](#)
- [Creating Disaster Recovery Mechanisms Using Amazon Route 53](#)

Related examples:

- [Disaster Recovery on AWS](#)
- [Elastic Disaster Recovery on AWS](#)

REL11-BP03 Automate healing on all layers

Upon detection of a failure, use automated capabilities to perform actions to remediate. Degradations may be automatically healed through internal service mechanisms or require resources to be restarted or removed through remediation actions.

For self-managed applications and cross-Region healing, recovery designs and automated healing processes can be pulled from [existing best practices](#).

The ability to restart or remove a resource is an important tool to remediate failures. A best practice is to make services stateless where possible. This prevents loss of data or availability on resource restart. In the cloud, you can (and generally should) replace the entire resource (for example, a compute instance or serverless function) as part of the restart. The restart itself is a simple and reliable way to recover from failure. Many different types of failures occur in workloads. Failures can occur in hardware, software, communications, and operations.

Restarting or retrying also applies to network requests. Apply the same recovery approach to both a network timeout and a dependency failure where the dependency returns an error. Both events have a similar effect on the system, so rather than attempting to make either event a special case, apply a similar strategy of limited retry with exponential backoff and jitter. Ability to restart is a recovery mechanism featured in recovery-oriented computing and high availability cluster architectures.

Desired outcome: Automated actions are performed to remediate detection of a failure.

Common anti-patterns:

- Provisioning resources without autoscaling.
- Deploying applications in instances or containers individually.
- Deploying applications that cannot be deployed into multiple locations without using automatic recovery.
- Manually healing applications that automatic scaling and automatic recovery fail to heal.
- No automation to failover databases.
- Lack automated methods to reroute traffic to new endpoints.
- No storage replication.

Benefits of establishing this best practice: Automated healing can reduce your mean time to recovery and improve your availability.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Designs for Amazon EKS or other Kubernetes services should include both minimum and maximum replica or stateful sets and the minimum cluster and node group sizing. These mechanisms provide a minimum amount of continually-available processing resources while automatically remediating any failures using the Kubernetes control plane.

Design patterns that are accessed through a load balancer using compute clusters should leverage Auto Scaling groups. Elastic Load Balancing (ELB) automatically distributes incoming application traffic across multiple targets and virtual appliances in one or more Availability Zones (AZs).

Clustered compute-based designs that do not use load balancing should have their size designed for loss of at least one node. This will allow for the service to maintain itself running in potentially

reduced capacity while it's recovering a new node. Example services are Mongo, DynamoDB Accelerator, Amazon Redshift, Amazon EMR, Cassandra, Kafka, MSK-EC2, Couchbase, ELK, and Amazon OpenSearch Service. Many of these services can be designed with additional auto healing features. Some cluster technologies must generate an alert upon the loss a node triggering an automated or manual workflow to recreate a new node. This workflow can be automated using AWS Systems Manager to remediate issues quickly.

Amazon EventBridge can be used to monitor and filter for events such as CloudWatch alarms or changes in state in other AWS services. Based on event information, it can then invoke AWS Lambda, Systems Manager Automation, or other targets to run custom remediation logic on your workload. Amazon EC2 Auto Scaling can be configured to check for EC2 instance health. If the instance is in any state other than running, or if the system status is impaired, Amazon EC2 Auto Scaling considers the instance to be unhealthy and launches a replacement instance. For large-scale replacements (such as the loss of an entire Availability Zone), static stability is preferred for high availability.

Implementation steps

- Use Auto Scaling groups to deploy tiers in a workload. [Auto Scaling](#) can perform self-healing on stateless applications and add or remove capacity.
- For compute instances noted previously, use [load balancing](#) and choose the appropriate type of load balancer.
- Consider healing for Amazon RDS. With standby instances, configure for [auto failover](#) to the standby instance. For Amazon RDS Read Replica, automated workflow is required to make a read replica primary.
- Implement [automatic recovery on EC2 instances](#) that have applications deployed that cannot be deployed in multiple locations, and can tolerate rebooting upon failures. Automatic recovery can be used to replace failed hardware and restart the instance when the application is not capable of being deployed in multiple locations. The instance metadata and associated IP addresses are kept, as well as the [EBS volumes](#) and mount points to [Amazon Elastic File System](#) or [File Systems for Lustre](#) and [Windows](#). Using [AWS OpsWorks](#), you can configure automatic healing of EC2 instances at the layer level.
- Implement automated recovery using [AWS Step Functions](#) and [AWS Lambda](#) when you cannot use automatic scaling or automatic recovery, or when automatic recovery fails. When you cannot use automatic scaling, and either cannot use automatic recovery or automatic recovery fails, you can automate the healing using AWS Step Functions and AWS Lambda.

- [Amazon EventBridge](#) can be used to monitor and filter for events such as [CloudWatch alarms](#) or changes in state in other AWS services. Based on event information, it can then invoke AWS Lambda (or other targets) to run custom remediation logic on your workload.

Resources

Related best practices:

- [Availability Definition](#)
- [REL11-BP01 Monitor all components of the workload to detect failures](#)

Related documents:

- [How AWS Auto Scaling Works](#)
- [Amazon EC2 Automatic Recovery](#)
- [Amazon Elastic Block Store \(Amazon EBS\)](#)
- [Amazon Elastic File System \(Amazon EFS\)](#)
- [What is Amazon FSx for Lustre?](#)
- [What is Amazon FSx for Windows File Server?](#)
- [AWS OpsWorks: Using Auto Healing to Replace Failed Instances](#)
- [What is AWS Step Functions?](#)
- [What is AWS Lambda?](#)
- [What Is Amazon EventBridge?](#)
- [Using Amazon CloudWatch Alarms](#)
- [Amazon RDS Failover](#)
- [SSM - Systems Manager Automation](#)
- [Resilient Architecture Best Practices](#)

Related videos:

- [Automatically Provision and Scale OpenSearch Service](#)
- [Amazon RDS Failover Automatically](#)

Related examples:

- [Workshop on Auto Scaling](#)
- [Amazon RDS Failover Workshop](#)

Related tools:

- [CloudWatch](#)
- [CloudWatch X-Ray](#)

REL11-BP04 Rely on the data plane and not the control plane during recovery

Control planes provide the administrative APIs used to create, read and describe, update, delete, and list (CRUDL) resources, while data planes handle day-to-day service traffic. When implementing recovery or mitigation responses to potentially resiliency-impacting events, focus on using a minimal number of control plane operations to recover, rescale, restore, heal, or failover the service. Data plane action should supersede any activity during these degradation events.

For example, the following are all control plane actions: launching a new compute instance, creating block storage, and describing queue services. When you launch compute instances, the control plane has to perform multiple tasks like finding a physical host with capacity, allocating network interfaces, preparing local block storage volumes, generating credentials, and adding security rules. Control planes tend to be complicated orchestration.

Desired outcome: When a resource enters an impaired state, the system is capable of automatically or manually recovering by shifting traffic from impaired to healthy resources.

Common anti-patterns:

- Dependence on changing DNS records to re-route traffic.
- Dependence on control-plane scaling operations to replace impaired components due to insufficiently provisioned resources.
- Relying on extensive, multi service, multi-API control plane actions to remediate any category of impairment.

Benefits of establishing this best practice: Increased success rate for automated remediation can reduce your mean time to recovery and improve availability of the workload.

Level of risk exposed if this best practice is not established: Medium: For certain types of service degradations, control planes are affected. Dependencies on extensive use of the control plane for remediation may increase recovery time (RTO) and mean time to recovery (MTTR).

Implementation guidance

To limit data plane actions, assess each service for what actions are required to restore service.

Leverage Amazon Route 53 Application Recovery Controller to shift the DNS traffic. These features continually monitor your application's ability to recover from failures and allow you to control your application recovery across multiple AWS Regions, Availability Zones, and on premises.

Route 53 routing policies use the control plane, so do not rely on it for recovery. The Route 53 data planes answer DNS queries and perform and evaluate health checks. They are globally distributed and designed for a [100% availability service level agreement \(SLA\)](#).

The Route 53 management APIs and consoles where you create, update, and delete Route 53 resources run on control planes that are designed to prioritize the strong consistency and durability that you need when managing DNS. To achieve this, the control planes are located in a single Region: US East (N. Virginia). While both systems are built to be very reliable, the control planes are not included in the SLA. There could be rare events in which the data plane's resilient design allows it to maintain availability while the control planes do not. For disaster recovery and failover mechanisms, use data plane functions to provide the best possible reliability.

For Amazon EC2, use static stability designs to limit control plane actions. Control plane actions include the scaling up of resources individually or using Auto Scaling groups (ASG). For the highest levels of resilience, provision sufficient capacity in the cluster used for failover. If this capacity threshold must be limited, set throttles on the overall end-to-end system to safely limit the total traffic reaching the limited set of resources.

For services like Amazon DynamoDB, Amazon API Gateway, load balancers, and AWS Lambda serverless, using those services leverages the data plane. However, creating new functions, load balancers, API gateways, or DynamoDB tables is a control plane action and should be completed before the degradation as preparation for an event and rehearsal of failover actions. For Amazon RDS, data plane actions allow for access to data.

For more information about data planes, control planes, and how AWS builds services to meet high availability targets, see [Static stability using Availability Zones](#).

Understand which operations are on the data plane and which are on the control plane.

Implementation steps

For each workload that needs to be restored after a degradation event, evaluate the failover runbook, high availability design, auto healing design, or HA resource restoration plan. Identity each action that might be considered a control plane action.

Consider changing the control action to a data plane action:

- Auto Scaling (control plane) compared to pre-scaled Amazon EC2 resources (data plane)
- Migrate to Lambda and its scaling methods (data plane) or Amazon EC2 and ASG (control plane)
- Assess any designs using Kubernetes and the nature of the control plane actions. Adding pods is a data plane action in Kubernetes. Actions should be limited to adding pods and not adding nodes. Using [over-provisioned nodes](#) is the preferred method to limit control plane actions

Consider alternate approaches that allow for data plane actions to affect the same remediation.

- Route 53 Record change (control plane) or Route 53 ARC (data plane)
- [Route 53 Health checks for more automated updates](#)

Consider some services in a secondary Region, if the service is mission critical, to allow for more control plane and data plane actions in an unaffected Region.

- Amazon EC2 Auto Scaling or Amazon EKS in a primary Region compared to Amazon EC2 Auto Scaling or Amazon EKS in a secondary Region and routing traffic to secondary Region (control plane action)
- Make read replica in secondary primary or attempting same action in primary Region (control plane action)

Resources

Related best practices:

- [Availability Definition](#)
- [REL11-BP01 Monitor all components of the workload to detect failures](#)

Related documents:

- [APN Partner: partners that can help with automation of your fault tolerance](#)
- [AWS Marketplace: products that can be used for fault tolerance](#)
- [Amazon Builders' Library: Avoiding overload in distributed systems by putting the smaller service in control](#)
- [Amazon DynamoDB API \(control plane and data plane\)](#)
- [AWS Lambda Executions \(split into the control plane and the data plane\)](#)
- [AWS Elemental MediaStore Data Plane](#)
- [Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 1: Single-Region stack](#)
- [Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 2: Multi-Region stack](#)
- [Creating Disaster Recovery Mechanisms Using Amazon Route 53](#)
- [What is Route 53 Application Recovery Controller](#)
- [Kubernetes Control Plane and data plane](#)

Related videos:

- [Back to Basics - Using Static Stability](#)
- [Building resilient multi-site workloads using AWS global services](#)

Related examples:

- [Introducing Amazon Route 53 Application Recovery Controller](#)
- [Amazon Builders' Library: Avoiding overload in distributed systems by putting the smaller service in control](#)
- [Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 1: Single-Region stack](#)
- [Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 2: Multi-Region stack](#)
- [Static stability using Availability Zones](#)

Related tools:

- [Amazon CloudWatch](#)
- [AWS X-Ray](#)

REL11-BP05 Use static stability to prevent bimodal behavior

Workloads should be statically stable and only operate in a single normal mode. Bimodal behavior is when your workload exhibits different behavior under normal and failure modes.

For example, you might try and recover from an Availability Zone failure by launching new instances in a different Availability Zone. This can result in a bimodal response during a failure mode. You should instead build workloads that are statically stable and operate within only one mode. In this example, those instances should have been provisioned in the second Availability Zone before the failure. This static stability design verifies that the workload only operates in a single mode.

Desired outcome: Workloads do not exhibit bimodal behavior during normal and failure modes.

Common anti-patterns:

- Assuming resources can always be provisioned regardless of the failure scope.
- Trying to dynamically acquire resources during a failure.
- Not provisioning adequate resources across zones or Regions until a failure occurs.
- Considering static stable designs for compute resources only.

Benefits of establishing this best practice: Workloads running with statically stable designs are capable of having predictable outcomes during normal and failure events.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Bimodal behavior occurs when your workload exhibits different behavior under normal and failure modes (for example, relying on launching new instances if an Availability Zone fails). An example of bimodal behavior is when stable Amazon EC2 designs provision enough instances in each Availability Zone to handle the workload load if one AZ were removed. Elastic Load Balancing or Amazon Route 53 health would check to shift a load away from the impaired instances. After traffic has shifted, use AWS Auto Scaling to asynchronously replace instances from the failed zone and

launch them in the healthy zones. Static stability for compute deployment (such as EC2 instances or containers) results in the highest reliability.



Static stability of EC2 instances across Availability Zones

This must be weighed against the cost for this model and the business value of maintaining the workload under all resilience cases. It's less expensive to provision less compute capacity and rely on launching new instances in the case of a failure, but for large-scale failures (such as an Availability Zone or Regional impairment), this approach is less effective because it relies on both an operational plane, and sufficient resources being available in the unaffected zones or Regions.

Your solution should also weigh reliability against the costs needs for your workload. Static stability architectures apply to a variety of architectures including compute instances spread across Availability Zones, database read replica designs, Kubernetes (Amazon EKS) cluster designs, and multi-Region failover architectures.

It is also possible to implement a more statically stable design by using more resources in each zone. By adding more zones, you reduce the amount of additional compute you need for static stability.

An example of bimodal behavior would be a network timeout that could cause a system to attempt to refresh the configuration state of the entire system. This would add an unexpected load to another component and might cause it to fail, resulting in other unexpected consequences. This negative feedback loop impacts the availability of your workload. Instead, you can build systems that are statically stable and operate in only one mode. A statically stable design would do constant work and always refresh the configuration state on a fixed cadence. When a call fails, the workload would use the previously cached value and initiate an alarm.

Another example of bimodal behavior is allowing clients to bypass your workload cache when failures occur. This might seem to be a solution that accommodates client needs but it can significantly change the demands on your workload and is likely to result in failures.

Assess critical workloads to determine what workloads require this type of resilience design. For those that are deemed critical, each application component must be reviewed. Example types of services that require static stability evaluations are:

- **Compute:** Amazon EC2, EKS-EC2, ECS-EC2, EMR-EC2
- **Databases:** Amazon Redshift, Amazon RDS, Amazon Aurora
- **Storage:** Amazon S3 (Single Zone), Amazon EFS (mounts), Amazon FSx (mounts)
- **Load balancers:** Under certain designs

Implementation steps

- Build systems that are statically stable and operate in only one mode. In this case, provision enough instances in each Availability Zone or Region to handle the workload capacity if one Availability Zone or Region were removed. A variety of services can be used for routing to healthy resources, such as:
 - [Cross Region DNS Routing](#)
 - [MRAP Amazon S3 MultiRegion Routing](#)
 - [AWS Global Accelerator](#)
 - [Amazon Route 53 Application Recovery Controller](#)
- Configure [database read replicas](#) to account for the loss of a single primary instance or a read replica. If traffic is being served by read replicas, the quantity in each Availability Zone and each Region should equate to the overall need in case of the zone or Region failure.
- Configure critical data in Amazon S3 storage that is designed to be statically stable for data stored in case of an Availability Zone failure. If [Amazon S3 One Zone-IA](#) storage class is used, this should not be considered statically stable, as the loss of that zone minimizes access to this stored data.
- [Load balancers](#) are sometimes configured incorrectly or by design to service a specific Availability Zone. In this case, the statically stable design might be to spread a workload across multiple AZs in a more complex design. The original design may be used to reduce interzone traffic for security, latency, or cost reasons.

Resources

Related Well-Architected best practices:

- [Availability Definition](#)
- [REL11-BP01 Monitor all components of the workload to detect failures](#)
- [REL11-BP04 Rely on the data plane and not the control plane during recovery](#)

Related documents:

- [Minimizing Dependencies in a Disaster Recovery Plan](#)
- [The Amazon Builders' Library: Static stability using Availability Zones](#)
- [Fault Isolation Boundaries](#)
- [Static stability using Availability Zones](#)
- [Multi-Zone RDS](#)
- [Minimizing Dependencies in a Disaster Recovery Plan](#)
- [Cross Region DNS Routing](#)
- [MRAP Amazon S3 MultiRegion Routing](#)
- [AWS Global Accelerator](#)
- [Route 53 ARC](#)
- [Single Zone Amazon S3](#)
- [Cross Zone Load Balancing](#)

Related videos:

- [Static stability in AWS: AWS re:Invent 2019: Introducing The Amazon Builders' Library \(DOP328\)](#)

Related examples:

- [The Amazon Builders' Library: Static stability using Availability Zones](#)

REL11-BP06 Send notifications when events impact availability

Notifications are sent upon the detection of thresholds breached, even if the event causing the issue was automatically resolved.

Automated healing allows your workload to be reliable. However, it can also obscure underlying problems that need to be addressed. Implement appropriate monitoring and events so that you can detect patterns of problems, including those addressed by auto healing, so that you can resolve root cause issues.

Resilient systems are designed so that degradation events are immediately communicated to the appropriate teams. These notifications should be sent through one or many communication channels.

Desired outcome: Alerts are immediately sent to operations teams when thresholds are breached, such as error rates, latency, or other critical key performance indicator (KPI) metrics, so that these issues are resolved as soon as possible and user impact is avoided or minimized.

Common anti-patterns:

- Sending too many alarms.
- Sending alarms that are not actionable.
- Setting alarm thresholds too high (over sensitive) or too low (under sensitive).
- Not sending alarms for external dependencies.
- Not considering gray failures when designing monitoring and alarms.
- Performing healing automation, but not notifying the appropriate team that healing was needed.

Benefits of establishing this best practice: Notifications of recovery make operational and business teams aware of service degradations so that they can react immediately to minimize both mean time to detect (MTTD) and mean time to repair (MTTR). Notifications of recovery events also assure that you don't ignore problems that occur infrequently.

Level of risk exposed if this best practice is not established: Medium. Failure to implement appropriate monitoring and events notification mechanisms can result in failure to detect patterns of problems, including those addressed by auto healing. A team will only be made aware of system degradation when users contact customer service or by chance.

Implementation guidance

When defining a monitoring strategy, a triggered alarm is a common event. This event would likely contain an identifier for the alarm, the alarm state (such as IN ALARM or OK), and details

of what triggered it. In many cases, an alarm event should be detected and an email notification sent. This is an example of an action on an alarm. Alarm notification is critical in observability, as it informs the right people that there is an issue. However, when action on events mature in your observability solution, it can automatically remediate the issue without the need for human intervention.

Once KPI-monitoring alarms have been established, alerts should be sent to appropriate teams when thresholds are exceeded. Those alerts may also be used to trigger automated processes that will attempt to remediate the degradation.

For more complex threshold monitoring, composite alarms should be considered. Composite alarms use a number of KPI-monitoring alarms to create an alert based on operational business logic. CloudWatch Alarms can be configured to send emails, or to log incidents in third-party incident tracking systems using Amazon SNS integration or Amazon EventBridge.

Implementation steps

Create various types of alarms based on how the workloads are monitored, such as:

- Application alarms are used to detect when any part of your workload is not working properly.
- [Infrastructure alarms](#) indicate when to scale resources. Alarms can be visually displayed on dashboards, send alerts through Amazon SNS or email, and work with Auto Scaling to scale workload resources in or out.
- Simple [static alarms](#) can be created to monitor when a metric breaches a static threshold for a specified number of evaluation periods.
- [Composite alarms](#) can account for complex alarms from multiple sources.
- Once the alarm has been created, create appropriate notification events. You can directly invoke an [Amazon SNS API](#) to send notifications and link any automation for remediation or communication.
- Integrate [Amazon Health Aware](#) monitoring to allow for monitoring visibility to AWS resources that might have degradations. For business essential workloads, this solution provides access to proactive and real-time alerts for AWS services.

Resources

Related Well-Architected best practices:

- [Availability Definition](#)

Related documents:

- [Creating a CloudWatch Alarm Based on a Static Threshold](#)
- [What Is Amazon EventBridge?](#)
- [What is Amazon Simple Notification Service?](#)
- [Publishing Custom Metrics](#)
- [Using Amazon CloudWatch Alarms](#)
- [Amazon Health Aware \(AHA\)](#)
- [Setup CloudWatch Composite alarms](#)
- [What's new in AWS Observability at re:Invent 2022](#)

Related tools:

- [CloudWatch](#)
- [CloudWatch X-Ray](#)

REL11-BP07 Architect your product to meet availability targets and uptime service level agreements (SLAs)

Architect your product to meet availability targets and uptime service level agreements (SLAs). If you publish or privately agree to availability targets or uptime SLAs, verify that your architecture and operational processes are designed to support them.

Desired outcome: Each application has a defined target for availability and SLA for performance metrics, which can be monitored and maintained in order to meet business outcomes.

Common anti-patterns:

- Designing and deploying workload's without setting any SLAs.
- SLA metrics are set too high without rationale or business requirements.
- Setting SLAs without taking into account for dependencies and their underlying SLA.
- Application designs are created without considering the Shared Responsibility Model for Resilience.

Benefits of establishing this best practice: Designing applications based on key resiliency targets helps you meet business objectives and customer expectations. These objectives help drive the application design process that evaluates different technologies and considers various tradeoffs.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Application designs have to account for a diverse set of requirements that are derived from business, operational, and financial objectives. Within the operational requirements, workloads need to have specific resilience metric targets so they can be properly monitored and supported. Resilience metrics should not be set or derived after deploying the workload. They should be defined during the design phase and help guide various decisions and tradeoffs.

- Every workload should have its own set of resilience metrics. Those metrics may be different from other business applications.
- Reducing dependencies can have a positive impact on availability. Each workload should consider its dependencies and their SLAs. In general, select dependencies with availability goals equal to or greater than the goals of your workload.
- Consider loosely coupled designs so your workload can operate correctly despite dependency impairment, where possible.
- Reduce control plane dependencies, especially during recovery or a degradation. Evaluate designs that are statically stable for mission critical workloads. Use resource sparing to increase the availability of those dependencies in a workload.
- Observability and instrumentation are critical for achieving SLAs by reducing Mean Time to Detection (MTTD) and Mean Time to Repair (MTTR).
- Less frequent failure (longer MTBF), shorter failure detection times (shorter MTTD), and shorter repair times (shorter MTTR) are the three factors that are used to improve availability in distributed systems.
- Establishing and meeting resilience metrics for a workload is foundational to any effective design. Those designs must factor in tradeoffs of design complexity, service dependencies, performance, scaling, and costs.

Implementation steps

- Review and document the workload design considering the following questions:
 - Where are control planes used in the workload?

- How does the workload implement fault tolerance?
- What are the design patterns for scaling, automatic scaling, redundancy, and highly available components?
- What are the requirements for data consistency and availability?
- Are there considerations for resource sparing or resource static stability?
- What are the service dependencies?
- Define SLA metrics based on the workload architecture while working with stakeholders.
Consider the SLAs of all dependencies used by the workload.
- Once the SLA target has been set, optimize the architecture to meet the SLA.
- Once the design is set that will meet the SLA, implement operational changes, process automation, and runbooks that also will have focus on reducing MTTD and MTTR.
- Once deployed, monitor and report on the SLA.

Resources

Related best practices:

- [REL03-BP01 Choose how to segment your workload](#)
- [REL10-BP01 Deploy the workload to multiple locations](#)
- [REL11-BP01 Monitor all components of the workload to detect failures](#)
- [REL11-BP03 Automate healing on all layers](#)
- [REL12-BP05 Test resiliency using chaos engineering](#)
- [REL13-BP01 Define recovery objectives for downtime and data loss](#)
- [Understanding workload health](#)

Related documents:

- [Availability with redundancy](#)
- [Reliability pillar - Availability](#)
- [Measuring availability](#)
- [AWS Fault Isolation Boundaries](#)
- [Shared Responsibility Model for Resiliency](#)
- [Static stability using Availability Zones](#)

- [AWS Service Level Agreements \(SLAs\)](#)
- [Guidance for Cell-based Architecture on AWS](#)
- [AWS infrastructure](#)
- [Advanced Multi-AZ Resiliency Patterns whitepaper](#)

Related services:

- [Amazon CloudWatch](#)
- [AWS Config](#)
- [AWS Trusted Advisor](#)

REL 12. How do you test reliability?

After you have designed your workload to be resilient to the stresses of production, testing is the only way to verify that it will operate as designed, and deliver the resiliency you expect.

Best practices

- [REL12-BP01 Use playbooks to investigate failures](#)
- [REL12-BP02 Perform post-incident analysis](#)
- [REL12-BP03 Test functional requirements](#)
- [REL12-BP04 Test scaling and performance requirements](#)
- [REL12-BP05 Test resiliency using chaos engineering](#)
- [REL12-BP06 Conduct game days regularly](#)

REL12-BP01 Use playbooks to investigate failures

Permit consistent and prompt responses to failure scenarios that are not well understood, by documenting the investigation process in playbooks. Playbooks are the predefined steps performed to identify the factors contributing to a failure scenario. The results from any process step are used to determine the next steps to take until the issue is identified or escalated.

The playbook is proactive planning that you must do, to be able to take reactive actions effectively. When failure scenarios not covered by the playbook are encountered in production, first address the issue (put out the fire). Then go back and look at the steps you took to address the issue and use these to add a new entry in the playbook.

Note that playbooks are used in response to specific incidents, while runbooks are used to achieve specific outcomes. Often, runbooks are used for routine activities and playbooks are used to respond to non-routine events.

Common anti-patterns:

- Planning to deploy a workload without knowing the processes to diagnose issues or respond to incidents.
- Unplanned decisions about which systems to gather logs and metrics from when investigating an event.
- Not retaining metrics and events long enough to be able to retrieve the data.

Benefits of establishing this best practice: Capturing playbooks ensures that processes can be consistently followed. Codifying your playbooks limits the introduction of errors from manual activity. Automating playbooks shortens the time to respond to an event by eliminating the requirement for team member intervention or providing them additional information when their intervention begins.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Use playbooks to identify issues. Playbooks are documented processes to investigate issues. Allow consistent and prompt responses to failure scenarios by documenting processes in playbooks. Playbooks must contain the information and guidance necessary for an adequately skilled person to gather applicable information, identify potential sources of failure, isolate faults, and determine contributing factors (perform post-incident analysis).
 - Implement playbooks as code. Perform your operations as code by scripting your playbooks to ensure consistency and limit reduce errors caused by manual processes. Playbooks can be composed of multiple scripts representing the different steps that might be necessary to identify the contributing factors to an issue. Runbook activities can be invoked or performed as part of playbook activities, or might prompt to run a playbook in response to identified events.
 - [Automate your operational playbooks with AWS Systems Manager](#)
 - [AWS Systems Manager Run Command](#)
 - [AWS Systems Manager Automation](#)
 - [What is AWS Lambda?](#)

- [What Is Amazon EventBridge?](#)
- [Using Amazon CloudWatch Alarms](#)

Resources

Related documents:

- [AWS Systems Manager Automation](#)
- [AWS Systems Manager Run Command](#)
- [Automate your operational playbooks with AWS Systems Manager](#)
- [Using Amazon CloudWatch Alarms](#)
- [Using Canaries \(Amazon CloudWatch Synthetics\)](#)
- [What Is Amazon EventBridge?](#)
- [What is AWS Lambda?](#)

Related examples:

- [Automating operations with Playbooks and Runbooks](#)

REL12-BP02 Perform post-incident analysis

This best practice was updated with new guidance on December 6, 2023.

Review customer-impacting events, and identify the contributing factors and preventative action items. Use this information to develop mitigations to limit or prevent recurrence. Develop procedures for prompt and effective responses. Communicate contributing factors and corrective actions as appropriate, tailored to target audiences. Have a method to communicate these causes to others as needed.

Assess why existing testing did not find the issue. Add tests for this case if tests do not already exist.

Desired outcome: Your teams have a consistent and agreed upon approach to handling post-incident analysis. One mechanism is the [correction of error \(COE\) process](#). The COE process helps

your teams identify, understand, and address the root causes for incidents, while also building mechanisms and guardrails to limit the probability of the same incident happening again.

Common anti-patterns:

- Finding contributing factors, but not continuing to look deeper for other potential problems and approaches to mitigate.
- Only identifying human error causes, and not providing any training or automation that could prevent human errors.
- Focus on assigning blame rather than understanding the root cause, creating a culture of fear and hindering open communication
- Failure to share insights, which keeps incident analysis findings within a small group and prevents others from benefiting from the lessons learned
- No mechanism to capture institutional knowledge, thereby losing valuable insights by not preserving the lessons-learned in the form of updated best practices and resulting in repeat incidents with the same or similar root cause

Benefits of establishing this best practice: Conducting post-incident analysis and sharing the results permits other workloads to mitigate the risk if they have implemented the same contributing factors, and allows them to implement the mitigation or automated recovery before an incident occurs.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Good post-incident analysis provides opportunities to propose common solutions for problems with architecture patterns that are used in other places in your systems.

A cornerstone of the COE process is documenting and addressing issues. It is recommended to define a standardized way to document critical root causes, and ensure they are reviewed and addressed. Assign clear ownership for the post-incident analysis process. Designate a responsible team or individual who will oversee incident investigations and follow-ups.

Encourage a culture that focuses on learning and improvement rather than assigning blame. Emphasize that the goal is to prevent future incidents, not to penalize individuals.

Develop well-defined procedures for conducting post-incident analyses. These procedures should outline the steps to be taken, the information to be collected, and the key questions to be

addressed during the analysis. Investigate incidents thoroughly, going beyond immediate causes to identify root causes and contributing factors. Use techniques like the [*five whys*](#) to delve deep into the underlying issues.

Maintain a repository of lessons learned from incident analyses. This institutional knowledge can serve as a reference for future incidents and prevention efforts. Share findings and insights from post-incident analyses, and consider holding open-invite post-incident review meetings to discuss lessons learned.

Implementation steps

- While conducting post-incident analysis, ensure the process is blame-free. This allows people involved in the incident to be dispassionate about the proposed corrective actions and promote honest self-assessment and collaboration across teams.
- Define a standardized way to document critical issues. An example structure for such document is as follows:
 - What happened?
 - What was the impact on customers and your business?
 - What was the root cause?
 - What data do you have to support this?
 - For example, metrics and graphs
 - What were the critical pillar implications, especially security?
 - When architecting workloads, you make trade-offs between pillars based upon your business context. These business decisions can drive your engineering priorities. You might optimize to reduce cost at the expense of reliability in development environments, or, for mission-critical solutions, you might optimize reliability with increased costs. Security is always job zero, as you have to protect your customers.
 - What lessons did you learn?
 - What corrective actions are you taking?
 - Action items
 - Related items
- Create well-defined standard operating procedures for conducting post-incident analyses.
- Set up a standardized incident reporting process. Document all incidents comprehensively, including the initial incident report, logs, communications, and actions taken during the incident.

- Remember that an incident does not require an outage. It could be a near-miss, or a system performing in an unexpected way while still fulfilling its business function.
- Continually improve your post-incident analysis process based on feedback and lessons learned.
- Capture key findings in a knowledge management system, and consider any patterns that should be added to developer guides or pre-deployment checklists.

Resources

Related documents:

- [Why you should develop a correction of error \(COE\)](#)

Related videos:

- [Amazon's approach to failing successfully](#)
- [AWS re:Invent 2021 - Amazon Builders' Library: Operational Excellence at Amazon](#)

REL12-BP03 Test functional requirements

Use techniques such as unit tests and integration tests that validate required functionality.

You achieve the best outcomes when these tests are run automatically as part of build and deployment actions. For instance, using AWS CodePipeline, developers commit changes to a source repository where CodePipeline automatically detects the changes. Those changes are built, and tests are run. After the tests are complete, the built code is deployed to staging servers for testing. From the staging server, CodePipeline runs more tests, such as integration or load tests. Upon the successful completion of those tests, CodePipeline deploys the tested and approved code to production instances.

Additionally, experience shows that synthetic transaction testing (also known as *canary testing*, but not to be confused with canary deployments) that can run and simulate customer behavior is among the most important testing processes. Run these tests constantly against your workload endpoints from diverse remote locations. Amazon CloudWatch Synthetics allows you to [create canaries](#) to monitor your endpoints and APIs.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Test functional requirements. These include unit tests and integration tests that validate required functionality.
 - [Use CodePipeline with AWS CodeBuild to test code and run builds](#)
 - [AWS CodePipeline Adds Support for Unit and Custom Integration Testing with AWS CodeBuild](#)
 - [Continuous Delivery and Continuous Integration](#)
 - [Using Canaries \(Amazon CloudWatch Synthetics\)](#)
 - [Software test automation](#)

Resources

Related documents:

- [APN Partner: partners that can help with implementation of a continuous integration pipeline](#)
- [AWS CodePipeline Adds Support for Unit and Custom Integration Testing with AWS CodeBuild](#)
- [AWS Marketplace: products that can be used for continuous integration](#)
- [Continuous Delivery and Continuous Integration](#)
- [Software test automation](#)
- [Use CodePipeline with AWS CodeBuild to test code and run builds](#)
- [Using Canaries \(Amazon CloudWatch Synthetics\)](#)

REL12-BP04 Test scaling and performance requirements

Use techniques such as load testing to validate that the workload meets scaling and performance requirements.

In the cloud, you can create a production-scale test environment on demand for your workload. If you run these tests on scaled down infrastructure, you must scale your observed results to what you think will happen in production. Load and performance testing can also be done in production if you are careful not to impact actual users, and tag your test data so it does not comingle with real user data and corrupt usage statistics or production reports.

With testing, ensure that your base resources, scaling settings, service quotas, and resiliency design operate as expected under load.

Level of risk exposed if this best practice is not established: High

Implementation guidance

- Test scaling and performance requirements. Perform load testing to validate that the workload meets scaling and performance requirements.
 - [Distributed Load Testing on AWS: simulate thousands of connected users](#)
 - [Apache JMeter](#)
 - Deploy your application in an environment identical to your production environment and run a load test.
 - Use infrastructure as code concepts to create an environment as similar to your production environment as possible.

Resources

Related documents:

- [Distributed Load Testing on AWS: simulate thousands of connected users](#)
- [Apache JMeter](#)

REL12-BP05 Test resiliency using chaos engineering

Run chaos experiments regularly in environments that are in or as close to production as possible to understand how your system responds to adverse conditions.

Desired outcome:

The resilience of the workload is regularly verified by applying chaos engineering in the form of fault injection experiments or injection of unexpected load, in addition to resilience testing that validates known expected behavior of your workload during an event. Combine both chaos engineering and resilience testing to gain confidence that your workload can survive component failure and can recover from unexpected disruptions with minimal to no impact.

Common anti-patterns:

- Designing for resiliency, but not verifying how the workload functions as a whole when faults occur.
- Never experimenting under real-world conditions and expected load.

- Not treating your experiments as code or maintaining them through the development cycle.
- Not running chaos experiments both as part of your CI/CD pipeline, as well as outside of deployments.
- Neglecting to use past post-incident analyses when determining which faults to experiment with.

Benefits of establishing this best practice: Injecting faults to verify the resilience of your workload allows you to gain confidence that the recovery procedures of your resilient design will work in the case of a real fault.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Chaos engineering provides your teams with capabilities to continually inject real world disruptions (simulations) in a controlled way at the service provider, infrastructure, workload, and component level, with minimal to no impact to your customers. It allows your teams to learn from faults and observe, measure, and improve the resilience of your workloads, as well as validate that alerts fire and teams get notified in the case of an event.

When performed continually, chaos engineering can highlight deficiencies in your workloads that, if left unaddressed, could negatively affect availability and operation.

Note

Chaos engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production. – [Principles of Chaos Engineering](#)

If a system is able to withstand these disruptions, the chaos experiment should be maintained as an automated regression test. In this way, chaos experiments should be performed as part of your systems development lifecycle (SDLC) and as part of your CI/CD pipeline.

To ensure that your workload can survive component failure, inject real world events as part of your experiments. For example, experiment with the loss of Amazon EC2 instances or failover of the primary Amazon RDS database instance, and verify that your workload is not impacted (or only minimally impacted). Use a combination of component faults to simulate events that may be caused by a disruption in an Availability Zone.

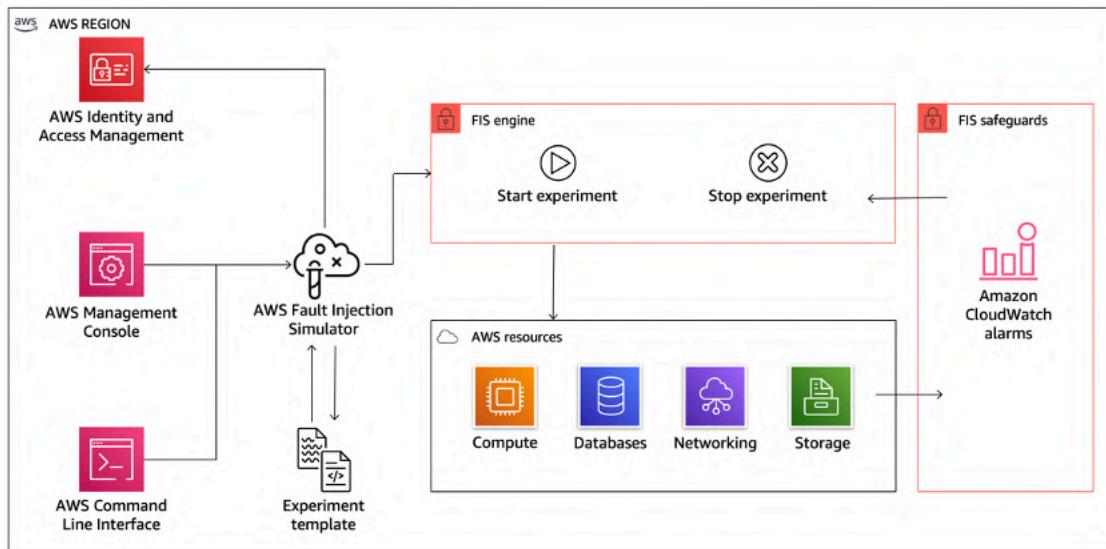
For application-level faults (such as crashes), you can start with stressors such as memory and CPU exhaustion.

To validate [fallback or failover mechanisms](#) for external dependencies due to intermittent network disruptions, your components should simulate such an event by blocking access to the third-party providers for a specified duration that can last from seconds to hours.

Other modes of degradation might cause reduced functionality and slow responses, often resulting in a disruption of your services. Common sources of this degradation are increased latency on critical services and unreliable network communication (dropped packets). Experiments with these faults, including networking effects such as latency, dropped messages, and DNS failures, could include the inability to resolve a name, reach the DNS service, or establish connections to dependent services.

Chaos engineering tools:

AWS Fault Injection Service (AWS FIS) is a fully managed service for running fault injection experiments that can be used as part of your CD pipeline, or outside of the pipeline. AWS FIS is a good choice to use during chaos engineering game days. It supports simultaneously introducing faults across different types of resources including Amazon EC2, Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and Amazon RDS. These faults include termination of resources, forcing failovers, stressing CPU or memory, throttling, latency, and packet loss. Since it is integrated with Amazon CloudWatch Alarms, you can set up stop conditions as guardrails to rollback an experiment if it causes unexpected impact.



AWS Fault Injection Service integrates with AWS resources to allow you to run fault injection experiments for your workloads.

There are also several third-party options for fault injection experiments. These include open-source tools such as [Chaos Toolkit](#), [Chaos Mesh](#), and [Litmus Chaos](#), as well as commercial options like Gremlin. To expand the scope of faults that can be injected on AWS, AWS FIS [integrates with Chaos Mesh and Litmus Chaos](#), allowing you to coordinate fault injection workflows among multiple tools. For example, you can run a stress test on a pod's CPU using Chaos Mesh or Litmus faults while terminating a randomly selected percentage of cluster nodes using AWS FIS fault actions.

Implementation steps

1. Determine which faults to use for experiments.

Assess the design of your workload for resiliency. Such designs (created using the best practices of the [Well-Architected Framework](#)) account for risks based on critical dependencies, past events, known issues, and compliance requirements. List each element of the design intended to maintain resilience and the faults it is designed to mitigate. For more information about creating such lists, see the [Operational Readiness Review whitepaper](#) which guides you on how to create a process to prevent reoccurrence of previous incidents. The Failure Modes and Effects Analysis (FMEA) process provides you with a framework for performing a component-level analysis of failures and how they impact your workload. FMEA is outlined in more detail by Adrian Cockcroft in [Failure Modes and Continuous Resilience](#).

2. Assign a priority to each fault.

Start with a coarse categorization such as high, medium, or low. To assess priority, consider frequency of the fault and impact of failure to the overall workload.

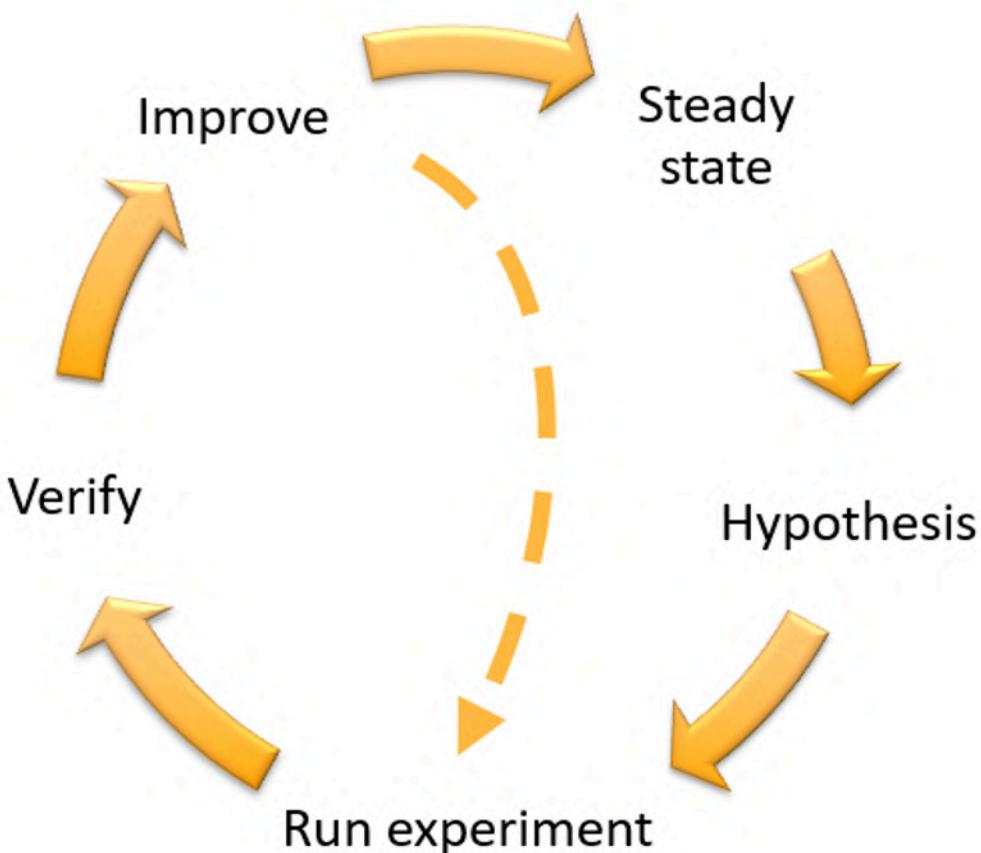
When considering frequency of a given fault, analyze past data for this workload when available. If not available, use data from other workloads running in a similar environment.

When considering impact of a given fault, the larger the scope of the fault, generally the larger the impact. Also consider the workload design and purpose. For example, the ability to access the source data stores is critical for a workload doing data transformation and analysis. In this case, you would prioritize experiments for access faults, as well as throttled access and latency insertion.

Post-incident analyses are a good source of data to understand both frequency and impact of failure modes.

Use the assigned priority to determine which faults to experiment with first and the order with which to develop new fault injection experiments.

3. For each experiment that you perform, follow the chaos engineering and continuous resilience flywheel in the following figure.



Chaos engineering and continuous resilience flywheel, using the scientific method by Adrian Hornsby.

- a. Define steady state as some measurable output of a workload that indicates normal behavior.

Your workload exhibits steady state if it is operating reliably and as expected. Therefore, validate that your workload is healthy before defining steady state. Steady state does not necessarily mean no impact to the workload when a fault occurs, as a certain percentage

in faults could be within acceptable limits. The steady state is your baseline that you will observe during the experiment, which will highlight anomalies if your hypothesis defined in the next step does not turn out as expected.

For example, a steady state of a payments system can be defined as the processing of 300 TPS with a success rate of 99% and round-trip time of 500 ms.

b. Form a hypothesis about how the workload will react to the fault.

A good hypothesis is based on how the workload is expected to mitigate the fault to maintain the steady state. The hypothesis states that given the fault of a specific type, the system or workload will continue steady state, because the workload was designed with specific mitigations. The specific type of fault and mitigations should be specified in the hypothesis.

The following template can be used for the hypothesis (but other wording is also acceptable):

 **Note**

If *specific fault* occurs, the *workload name* workload will *describe mitigating controls* to maintain *business or technical metric impact*.

For example:

- If 20% of the nodes in the Amazon EKS node-group are taken down, the Transaction Create API continues to serve the 99th percentile of requests in under 100 ms (steady state). The Amazon EKS nodes will recover within five minutes, and pods will get scheduled and process traffic within eight minutes after the initiation of the experiment. Alerts will fire within three minutes.
- If a single Amazon EC2 instance failure occurs, the order system's Elastic Load Balancing health check will cause the Elastic Load Balancing to only send requests to the remaining healthy instances while the Amazon EC2 Auto Scaling replaces the failed instance, maintaining a less than 0.01% increase in server-side (5xx) errors (steady state).
- If the primary Amazon RDS database instance fails, the Supply Chain data collection workload will failover and connect to the standby Amazon RDS database instance to maintain less than 1 minute of database read or write errors (steady state).

c. Run the experiment by injecting the fault.

An experiment should by default be fail-safe and tolerated by the workload. If you know that the workload will fail, do not run the experiment. Chaos engineering should be used to find known-unknowns or unknown-unknowns. *Known-unknowns* are things you are aware of but don't fully understand, and *unknown-unknowns* are things you are neither aware of nor fully understand. Experimenting against a workload that you know is broken won't provide you with new insights. Your experiment should be carefully planned, have a clear scope of impact, and provide a rollback mechanism that can be applied in case of unexpected turbulence. If your due-diligence shows that your workload should survive the experiment, move forward with the experiment. There are several options for injecting the faults. For workloads on AWS, [AWS FIS](#) provides many predefined fault simulations called [actions](#). You can also define custom actions that run in AWS FIS using [AWS Systems Manager documents](#).

We discourage the use of custom scripts for chaos experiments, unless the scripts have the capabilities to understand the current state of the workload, are able to emit logs, and provide mechanisms for rollbacks and stop conditions where possible.

An effective framework or toolset which supports chaos engineering should track the current state of an experiment, emit logs, and provide rollback mechanisms to support the controlled running of an experiment. Start with an established service like AWS FIS that allows you to perform experiments with a clearly defined scope and safety mechanisms that rollback the experiment if the experiment introduces unexpected turbulence. To learn about a wider variety of experiments using AWS FIS, also see the [Resilient and Well-Architected Apps with Chaos Engineering lab](#). Also, [AWS Resilience Hub](#) will analyze your workload and create experiments that you can choose to implement and run in AWS FIS.

Note

For every experiment, clearly understand the scope and its impact. We recommend that faults should be simulated first on a non-production environment before being run in production.

Experiments should run in production under real-world load using [canary deployments](#) that spin up both a control and experimental system deployment, where feasible. Running experiments during off-peak times is a good practice to mitigate potential impact when first experimenting in production. Also, if using actual customer traffic poses too much risk, you can run experiments using synthetic traffic on production infrastructure against the control

and experimental deployments. When using production is not possible, run experiments in pre-production environments that are as close to production as possible.

You must establish and monitor guardrails to ensure the experiment does not impact production traffic or other systems beyond acceptable limits. Establish stop conditions to stop an experiment if it reaches a threshold on a guardrail metric that you define. This should include the metrics for steady state for the workload, as well as the metric against the components into which you're injecting the fault. A [synthetic monitor](#) (also known as a user canary) is one metric you should usually include as a user proxy. [Stop conditions for AWS FIS](#) are supported as part of the experiment template, allowing up to five stop-conditions per template.

One of the principles of chaos is minimize the scope of the experiment and its impact:

While there must be an allowance for some short-term negative impact, it is the responsibility and obligation of the Chaos Engineer to ensure the fallout from experiments are minimized and contained.

A method to verify the scope and potential impact is to perform the experiment in a non-production environment first, verifying that thresholds for stop conditions activate as expected during an experiment and observability is in place to catch an exception, instead of directly experimenting in production.

When running fault injection experiments, verify that all responsible parties are well-informed. Communicate with appropriate teams such as the operations teams, service reliability teams, and customer support to let them know when experiments will be run and what to expect. Give these teams communication tools to inform those running the experiment if they see any adverse effects.

You must restore the workload and its underlying systems back to the original known-good state. Often, the resilient design of the workload will self-heal. But some fault designs or failed experiments can leave your workload in an unexpected failed state. By the end of the experiment, you must be aware of this and restore the workload and systems. With AWS FIS you can set a rollback configuration (also called a post action) within the action parameters. A post action returns the target to the state that it was in before the action was run. Whether automated (such as using AWS FIS) or manual, these post actions should be part of a playbook that describes how to detect and handle failures.

- d. Verify the hypothesis.

[Principles of Chaos Engineering](#) gives this guidance on how to verify steady state of your workload:

Focus on the measurable output of a system, rather than internal attributes of the system. Measurements of that output over a short period of time constitute a proxy for the system's steady state. The overall system's throughput, error rates, and latency percentiles could all be metrics of interest representing steady state behavior. By focusing on systemic behavior patterns during experiments, chaos engineering verifies that the system does work, rather than trying to validate how it works.

In our two previous examples, we include the steady state metrics of less than 0.01% increase in server-side (5xx) errors and less than one minute of database read and write errors.

The 5xx errors are a good metric because they are a consequence of the failure mode that a client of the workload will experience directly. The database errors measurement is good as a direct consequence of the fault, but should also be supplemented with a client impact measurement such as failed customer requests or errors surfaced to the client. Additionally, include a synthetic monitor (also known as a user canary) on any APIs or URIs directly accessed by the client of your workload.

e. Improve the workload design for resilience.

If steady state was not maintained, then investigate how the workload design can be improved to mitigate the fault, applying the best practices of the [AWS Well-Architected Reliability pillar](#). Additional guidance and resources can be found in the [AWS Builder's Library](#), which hosts articles about how to [improve your health checks](#) or [employ retries with backoff in your application code](#), among others.

After these changes have been implemented, run the experiment again (shown by the dotted line in the chaos engineering flywheel) to determine their effectiveness. If the verify step indicates the hypothesis holds true, then the workload will be in steady state, and the cycle continues.

4. Run experiments regularly.

A chaos experiment is a cycle, and experiments should be run regularly as part of chaos engineering. After a workload meets the experiment's hypothesis, the experiment should be automated to run continually as a regression part of your CI/CD pipeline. To learn how to do

this, see this blog on [how to run AWS FIS experiments using AWS CodePipeline](#). This lab on recurrent [AWS FIS experiments in a CI/CD pipeline](#) allows you to work hands-on.

Fault injection experiments are also a part of game days (see [REL12-BP06 Conduct game days regularly](#)). Game days simulate a failure or event to verify systems, processes, and team responses. The purpose is to actually perform the actions the team would perform as if an exceptional event happened.

5. Capture and store experiment results.

Results for fault injection experiments must be captured and persisted. Include all necessary data (such as time, workload, and conditions) to be able to later analyze experiment results and trends. Examples of results might include screenshots of dashboards, CSV dumps from your metric's database, or a hand-typed record of events and observations from the experiment. [Experiment logging with AWS FIS](#) can be part of this data capture.

Resources

Related best practices:

- [REL08-BP03 Integrate resiliency testing as part of your deployment](#)
- [REL13-BP03 Test disaster recovery implementation to validate the implementation](#)

Related documents:

- [What is AWS Fault Injection Service?](#)
- [What is AWS Resilience Hub?](#)
- [Principles of Chaos Engineering](#)
- [Chaos Engineering: Planning your first experiment](#)
- [Resilience Engineering: Learning to Embrace Failure](#)
- [Chaos Engineering stories](#)
- [Avoiding fallback in distributed systems](#)
- [Canary Deployment for Chaos Experiments](#)

Related videos:

- [AWS re:Invent 2020: Testing resiliency using chaos engineering \(ARC316\)](#)

- [AWS re:Invent 2019: Improving resiliency with chaos engineering \(DOP309-R1\)](#)
- [AWS re:Invent 2019: Performing chaos engineering in a serverless world \(CMY301\)](#)

Related examples:

- [Well-Architected lab: Level 300: Testing for Resiliency of Amazon EC2, Amazon RDS, and Amazon S3](#)
- [Chaos Engineering on AWS lab](#)
- [Resilient and Well-Architected Apps with Chaos Engineering lab](#)
- [Serverless Chaos lab](#)
- [Measure and Improve Your Application Resilience with AWS Resilience Hub lab](#)

Related tools:

- [AWS Fault Injection Service](#)
- AWS Marketplace: [Gremlin Chaos Engineering Platform](#)
- [Chaos Toolkit](#)
- [Chaos Mesh](#)
- [Litmus](#)

REL12-BP06 Conduct game days regularly

Use game days to regularly exercise your procedures for responding to events and failures as close to production as possible (including in production environments) with the people who will be involved in actual failure scenarios. Game days enforce measures to ensure that production events do not impact users.

Game days simulate a failure or event to test systems, processes, and team responses. The purpose is to actually perform the actions the team would perform as if an exceptional event happened. This will help you understand where improvements can be made and can help develop organizational experience in dealing with events. These should be conducted regularly so that your team builds *muscle memory* on how to respond.

After your design for resiliency is in place and has been tested in non-production environments, a game day is the way to ensure that everything works as planned in production. A game day, especially the first one, is an “all hands on deck” activity where engineers and operations are

all informed when it will happen, and what will occur. Runbooks are in place. Simulated events are run, including possible failure events, in the production systems in the prescribed manner, and impact is assessed. If all systems operate as designed, detection and self-healing will occur with little to no impact. However, if negative impact is observed, the test is rolled back and the workload issues are remedied, manually if necessary (using the runbook). Since game days often take place in production, all precautions should be taken to ensure that there is no impact on availability to your customers.

Common anti-patterns:

- Documenting your procedures, but never exercising them.
- Not including business decision makers in the test exercises.

Benefits of establishing this best practice: Conducting game days regularly ensures that all staff follows the policies and procedures when an actual incident occurs, and validates that those policies and procedures are appropriate.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Schedule game days to regularly exercise your runbooks and playbooks. Game days should involve everyone who would be involved in a production event: business owner, development staff, operational staff, and incident response teams.
 - Run your load or performance tests and then run your failure injection.
 - Look for anomalies in your runbooks and opportunities to exercise your playbooks.
 - If you deviate from your runbooks, refine the runbook or correct the behavior. If you exercise your playbook, identify the runbook that should have been used, or create a new one.

Resources

Related documents:

- [What is AWS GameDay?](#)

Related videos:

- [AWS re:Invent 2019: Improving resiliency with chaos engineering \(DOP309-R1\)](#)

Related examples:

- [AWS Well-Architected Labs - Testing Resiliency](#)

REL 13. How do you plan for disaster recovery (DR)?

Having backups and redundant workload components in place is the start of your DR strategy. [RTO and RPO are your objectives](#) for restoration of your workload. Set these based on business needs. Implement a strategy to meet these objectives, considering locations and function of workload resources and data. The probability of disruption and cost of recovery are also key factors that help to inform the business value of providing disaster recovery for a workload.

Best practices

- [REL13-BP01 Define recovery objectives for downtime and data loss](#)
- [REL13-BP02 Use defined recovery strategies to meet the recovery objectives](#)
- [REL13-BP03 Test disaster recovery implementation to validate the implementation](#)
- [REL13-BP04 Manage configuration drift at the DR site or Region](#)
- [REL13-BP05 Automate recovery](#)

REL13-BP01 Define recovery objectives for downtime and data loss

The workload has a recovery time objective (RTO) and recovery point objective (RPO).

Recovery Time Objective (RTO) is the maximum acceptable delay between the interruption of service and restoration of service. This determines what is considered an acceptable time window when service is unavailable.

Recovery Point Objective (RPO) is the maximum acceptable amount of time since the last data recovery point. This determines what is considered an acceptable loss of data between the last recovery point and the interruption of service.

RTO and RPO values are important considerations when selecting an appropriate Disaster Recovery (DR) strategy for your workload. These objectives are determined by the business, and then used by technical teams to select and implement a DR strategy.

Desired Outcome:

Every workload has an assigned RTO and RPO, defined based on business impact. The workload is assigned to a predefined tier, defining service availability and acceptable loss of data, with an associated RTO and RPO. If such tiering is not possible then this can be assigned bespoke per workload, with the intent to create tiers later. RTO and RPO are used as one of the primary considerations for selection of a disaster recovery strategy implementation for the workload. Additional considerations in picking a DR strategy are cost constraints, workload dependencies, and operational requirements.

For RTO, understand impact based on duration of an outage. Is it linear, or are there nonlinear implications? (for example, after four hours, you shut down a manufacturing line until the start of the next shift).

A disaster recovery matrix, like the following, can help you understand how workload criticality relates to recovery objectives. (Note that the actual values for the X and Y axes should be customized to your organization needs).

		Disaster Recovery Matrix				
		Recovery Point Objective				
		< 1 Minute	< 1 Hour	< 6 Hours	< 1 Day	+ 1 Day
Recovery Time Objective	< 10 Minutes	Critical	Critical	High	Medium	Medium
	< 2 Hours	Critical	High	Medium	Medium	Low
	< 8 Hours	High	Medium	Medium	Low	Low
	< 24 Hours	Medium	Medium	Low	Low	Low
	24 + Hours	Medium	Low	Low	Low	Low

Figure 16: Disaster recovery matrix

Common anti-patterns:

- No defined recovery objectives.
- Selecting arbitrary recovery objectives.
- Selecting recovery objectives that are too lenient and do not meet business objectives.
- Not understanding of the impact of downtime and data loss.
- Selecting unrealistic recovery objectives, such as zero time to recover and zero data loss, which may not be achievable for your workload configuration.

- Selecting recovery objectives more stringent than actual business objectives. This forces DR implementations that are costlier and more complicated than what the workload needs.
- Selecting recovery objectives incompatible with those of a dependent workload.
- Your recovery objectives do not consider regulatory compliance requirements.
- RTO and RPO defined for a workload, but never tested.

Benefits of establishing this best practice: Your recovery objectives for time and data loss are necessary to guide your DR implementation.

Level of risk exposed if this best practice is not established: High

Implementation guidance

For the given workload, you must understand the impact of downtime and lost data on your business. The impact generally grows larger with greater downtime or data loss, but the shape of this growth can differ based on the workload type. For example, you may be able to tolerate downtime for up to an hour with little impact, but after that impact quickly rises. Impact to business manifests in many forms including monetary cost (such as lost revenue), customer trust (and impact to reputation), operational issues (such as missing payroll or decreased productivity), and regulatory risk. Use the following steps to understand these impacts, and set RTO and RPO for your workload.

Implementation Steps

1. Determine your business stakeholders for this workload, and engage with them to implement these steps. Recovery objectives for a workload are a business decision. Technical teams then work with business stakeholders to use these objectives to select a DR strategy.

 **Note**

For steps 2 and 3, you can use the [the section called “Implementation worksheet”](#).

2. Gather the necessary information to make a decision by answering the questions below.
3. Do you have categories or tiers of criticality for workload impact in your organization?
 - a. If yes, assign this workload to a category
 - b. If no, then establish these categories. Create five or fewer categories and refine the range of your recovery time objective for each one. Example categories include: critical, high, medium,

- low. To understand how workloads map to categories, consider whether the workload is mission critical, business important, or non-business driving.
- c. Set workload RTO and RPO based on category. Always choose a category more strict (lower RTO and RPO) than the raw values calculated entering this step. If this results in an unsuitably large change in value, then consider creating a new category.
 4. Based on these answers, assign RTO and RPO values to the workload. This can be done directly, or by assigning the workload to a predefined tier of service.
 5. Document the disaster recovery plan (DRP) for this workload, which is a part of your organization's [business continuity plan \(BCP\)](#), in a location accessible to the workload team and stakeholders
 - a. Record the RTO and RPO, and the information used to determine these values. Include the strategy used for evaluating workload impact to the business
 - b. Record other metrics besides RTO and RPO are you tracking or plan to track for disaster recovery objectives
 - c. You will add details of your DR strategy and runbook to this plan when you create these.
 6. By looking up the workload criticality in a matrix such as that in Figure 15, you can begin to establish predefined tiers of service defined for your organization.
 7. After you have implemented a DR strategy (or a proof of concept for a DR strategy) as per [the section called "REL13-BP02 Use defined recovery strategies to meet the recovery objectives"](#), test this strategy to determine workload actual RTC (Recovery Time Capability) and RPC (Recovery Point Capability). If these do not meet the target recovery objectives, then either work with your business stakeholders to adjust those objectives, or make changes to the DR strategy is possible to meet target objectives.

Primary questions

1. What is the maximum time the workload can be down before severe impact to the business is incurred
 - a. Determine the monetary cost (direct financial impact) to the business per minute if workload is disrupted.
 - b. Consider that impact is not always linear. Impact can be limited at first, and then increase rapidly past a critical point in time.
2. What is the maximum amount of data that can be lost before severe impact to the business is incurred

- a. Consider this value for your most critical data store. Identify the respective criticality for other data stores.
 - b. Can workload data be recreated if lost? If this is operationally easier than backup and restore, then choose RPO based on the criticality of the source data used to recreate the workload data.
3. What are the recovery objectives and availability expectations of workloads that this one depends on (downstream), or workloads that depend on this one (upstream)?
- a. Choose recovery objectives that allow this workload to meet the requirements of upstream dependencies
 - b. Choose recovery objectives that are achievable given the recovery capabilities of downstream dependencies. Non-critical downstream dependencies (ones you can “work around”) can be excluded. Or, work with critical downstream dependencies to improve their recovery capabilities where necessary.

Additional questions

Consider these questions, and how they may apply to this workload:

4. Do you have different RTO and RPO depending on the type of outage (Region vs. AZ, etc.)?
 5. Is there a specific time (seasonality, sales events, product launches) when your RTO/RPO may change? If so, what is the different measurement and time boundary?
 6. How many customers will be impacted if workload is disrupted?
 7. What is the impact to reputation if workload is disrupted?
 8. What other operational impacts may occur if workload is disrupted? For example, impact to employee productivity if email systems are unavailable, or if Payroll systems are unable to submit transactions.
 9. How does workload RTO and RPO align with Line of Business and Organizational DR Strategy?
10. Are there internal contractual obligations for providing a service? Are there penalties for not meeting them?
11. What are the regulatory or compliance constraints with the data?

Implementation worksheet

You can use this worksheet for implementation steps 2 and 3. You may adjust this worksheet to suit your specific needs, such as adding additional questions.

Step 2: Primary questions	Applies to workload?	workload RTO	workload RPO	RTO adjust.	RPO adjust.	Instructions
[1] maximum time the workload can be down						measured in time from start of outage to recovery
[2] maximum amount of data that can be lost						measured in time since last known good restorable dataset
[3a] upstream dependencies						enter the most strict upstream recovery objectives
[3b] downstream dependencies						enter the least strict downstream recovery objectives
[3a] reconciled upstream dependencies						If upstream value is less than current values and downstream value greater, then work with dependencies to reconcile and enter reconciled values here
[3b] reconciled downstream dependencies						lower values to meet upstream dependencies or raise them based on downstream dependency capabilities
[3] dependencies						
Step 2: Additional questions						Indicate if question applies. If it does not apply then skip it
Base RTO/RPO						Carry RTO and RPO values from above down to here
[4] type of outage	[]Y/[]N					Enter recovery objectives for event type with strictest requirements
[5] specific time-based objectives	[]Y/[]N					Enter recovery objectives for times with the strictest requirements
[6] customers disrupted	[]Y/[]N					Graph customers impacted as a function of time down or data lost. Use that to enter the maximum RTO and RPO permissible based on customer impact
[7] reputation impact	[]Y/[]N					Work with the business to determine maximum RTO and RPO based on impact to reputation
[8] operational impact	[]Y/[]N					Enter maximum RTO and RPO based on operational impact
[9] organizational alignment	[]Y/[]N					Enter maximum RTO and RPO for workloads of this type as per LOB and organizational requirements
[10] contractual obligations	[]Y/[]N					Enter maximum RTO and RPO based on contractual obligations
[11] regulatory compliance	[]Y/[]N					Enter maximum RTO and RPO based on applicable regulatory compliance
target based on additional questions						Take the minimum value (stricter value) from Q's 4-11 and enter it here
adjusted target						If the objectives on the above line cannot be accommodated, work with stakeholders to loosen constraints, and enter new minimum here
Adjusted RTO/RPO						Enter base RPO/RTO values, or adjusted target, whichever is lower
Step 3						
Map to predefined category or tier						Adjust both values to downward (more strict) to align to nearest defined tier

Worksheet

Level of effort for the Implementation Plan: Low

Resources

Related Best Practices:

- [the section called "REL09-BP04 Perform periodic recovery of the data to verify backup integrity and processes"](#)
- [the section called "REL13-BP02 Use defined recovery strategies to meet the recovery objectives"](#)
- [the section called "REL13-BP03 Test disaster recovery implementation to validate the implementation"](#)

Related documents:

- [AWS Architecture Blog: Disaster Recovery Series](#)
- [Disaster Recovery of Workloads on AWS: Recovery in the Cloud \(AWS Whitepaper\)](#)
- [Managing resiliency policies with AWS Resilience Hub](#)

- [APN Partner: partners that can help with disaster recovery](#)
- [AWS Marketplace: products that can be used for disaster recovery](#)

Related videos:

- [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications \(ARC209-R2\)](#)
- [Disaster Recovery of Workloads on AWS](#)

REL13-BP02 Use defined recovery strategies to meet the recovery objectives

Define a disaster recovery (DR) strategy that meets your workload's recovery objectives. Choose a strategy such as backup and restore, standby (active/passive), or active/active.

Desired outcome: For each workload, there is a defined and implemented DR strategy that allows the workload to achieve DR objectives. DR strategies between workloads make use of reusable patterns (such as the strategies previously described),

Common anti-patterns:

- Implementing inconsistent recovery procedures for workloads with similar DR objectives.
- Leaving the DR strategy to be implemented ad-hoc when a disaster occurs.
- Having no plan for disaster recovery.
- Dependency on control plane operations during recovery.

Benefits of establishing this best practice:

- Using defined recovery strategies allows you to use common tooling and test procedures.
- Using defined recovery strategies improves knowledge sharing between teams and implementation of DR on the workloads they own.

Level of risk exposed if this best practice is not established: High. Without a planned, implemented, and tested DR strategy, you are unlikely to achieve recovery objectives in the event of a disaster.

Implementation guidance

A DR strategy relies on the ability to stand up your workload in a recovery site if your primary location becomes unable to run the workload. The most common recovery objectives are RTO and RPO, as discussed in [REL13-BP01 Define recovery objectives for downtime and data loss](#).

A DR strategy across multiple Availability Zones (AZs) within a single AWS Region, can provide mitigation against disaster events like fires, floods, and major power outages. If it is a requirement to implement protection against an unlikely event that prevents your workload from being able to run in a given AWS Region, you can use a DR strategy that uses multiple Regions.

When architecting a DR strategy across multiple Regions, you should choose one of the following strategies. They are listed in increasing order of cost and complexity, and decreasing order of RTO and RPO. *Recovery Region* refers to an AWS Region other than the primary one used for your workload.

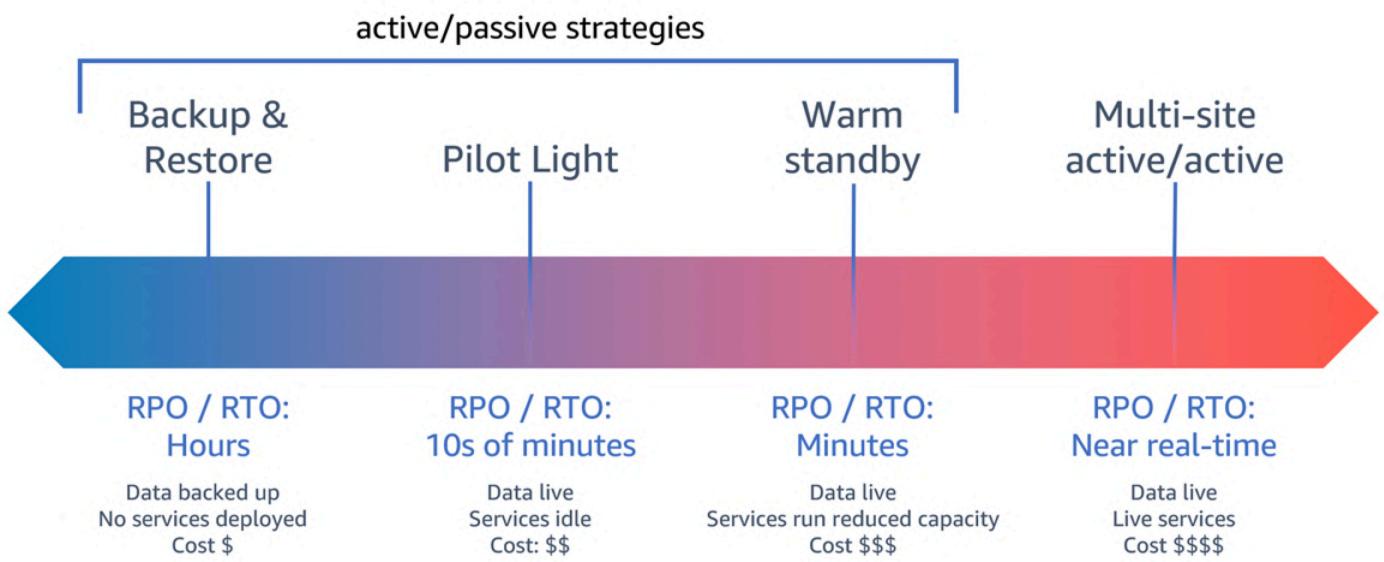


Figure 17: Disaster recovery (DR) strategies

- **Backup and restore** (RPO in hours, RTO in 24 hours or less): Back up your data and applications into the recovery Region. Using automated or continuous backups will permit point in time recovery (PITR), which can lower RPO to as low as 5 minutes in some cases. In the event of a disaster, you will deploy your infrastructure (using infrastructure as code to reduce RTO), deploy your code, and restore the backed-up data to recover from a disaster in the recovery Region.

- **Pilot light** (RPO in minutes, RTO in tens of minutes): Provision a copy of your core workload infrastructure in the recovery Region. Replicate your data into the recovery Region and create backups of it there. Resources required to support data replication and backup, such as databases and object storage, are always on. Other elements such as application servers or serverless compute are not deployed, but can be created when needed with the necessary configuration and application code.
- **Warm standby** (RPO in seconds, RTO in minutes): Maintain a scaled-down but fully functional version of your workload always running in the recovery Region. Business-critical systems are fully duplicated and are always on, but with a scaled down fleet. Data is replicated and live in the recovery Region. When the time comes for recovery, the system is scaled up quickly to handle the production load. The more scaled-up the warm standby is, the lower RTO and control plane reliance will be. When fully scales this is known as *hot standby*.
- **Multi-Region (multi-site) active-active** (RPO near zero, RTO potentially zero): Your workload is deployed to, and actively serving traffic from, multiple AWS Regions. This strategy requires you to synchronize data across Regions. Possible conflicts caused by writes to the same record in two different regional replicas must be avoided or handled, which can be complex. Data replication is useful for data synchronization and will protect you against some types of disaster, but it will not protect you against data corruption or destruction unless your solution also includes options for point-in-time recovery.

Note

The difference between pilot light and warm standby can sometimes be difficult to understand. Both include an environment in your recovery Region with copies of your primary region assets. The distinction is that pilot light cannot process requests without additional action taken first, while warm standby can handle traffic (at reduced capacity levels) immediately. Pilot light will require you to turn on servers, possibly deploy additional (non-core) infrastructure, and scale up, while warm standby only requires you to scale up (everything is already deployed and running). Choose between these based on your RTO and RPO needs.

When cost is a concern, and you wish to achieve a similar RPO and RTO objectives as defined in the warm standby strategy, you could consider cloud native solutions, like AWS Elastic Disaster Recovery, that take the pilot light approach and offer improved RPO and RTO targets.

Implementation steps

1. Determine a DR strategy that will satisfy recovery requirements for this workload.

Choosing a DR strategy is a trade-off between reducing downtime and data loss (RTO and RPO) and the cost and complexity of implementing the strategy. You should avoid implementing a strategy that is more stringent than it needs to be, as this incurs unnecessary costs.

For example, in the following diagram, the business has determined their maximum permissible RTO as well as the limit of what they can spend on their service restoration strategy. Given the business' objectives, the DR strategies pilot light or warm standby will satisfy both the RTO and the cost criteria.

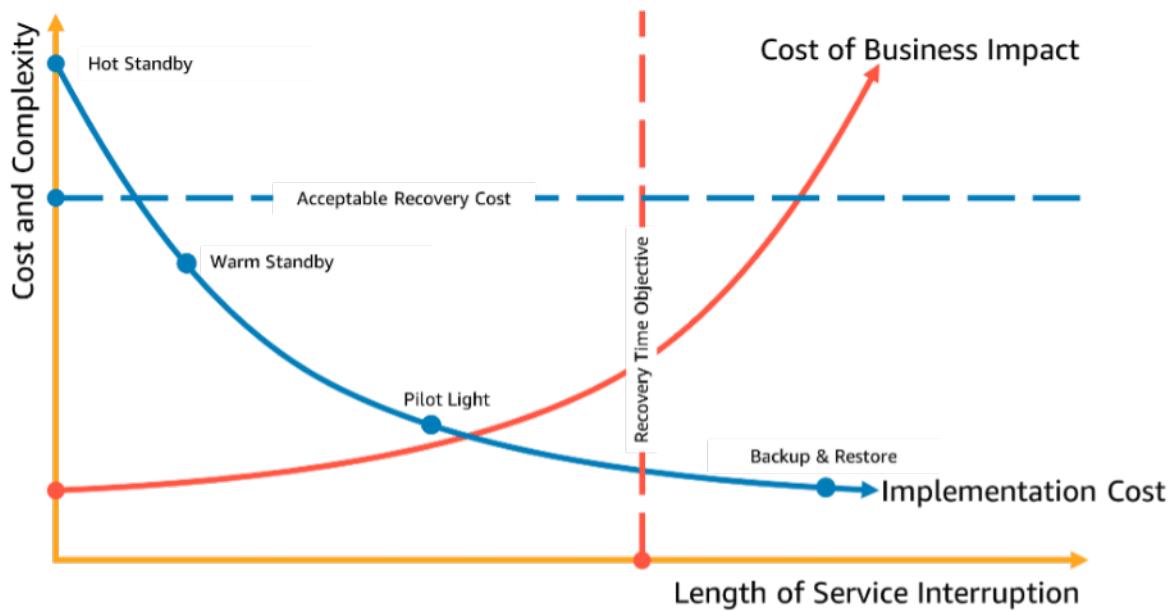


Figure 18: Choosing a DR strategy based on RTO and cost

To learn more, see [Business Continuity Plan \(BCP\)](#).

2. Review the patterns for how the selected DR strategy can be implemented.

This step is to understand how you will implement the selected strategy. The strategies are explained using AWS Regions as the primary and recovery sites. However, you can also choose to use Availability Zones within a single Region as your DR strategy, which makes use of elements of multiple of these strategies.

In the following steps, you can apply the strategy to your specific workload.

Backup and restore

Backup and restore is the least complex strategy to implement, but will require more time and effort to restore the workload, leading to higher RTO and RPO. It is a good practice to always make backups of your data, and copy these to another site (such as another AWS Region).

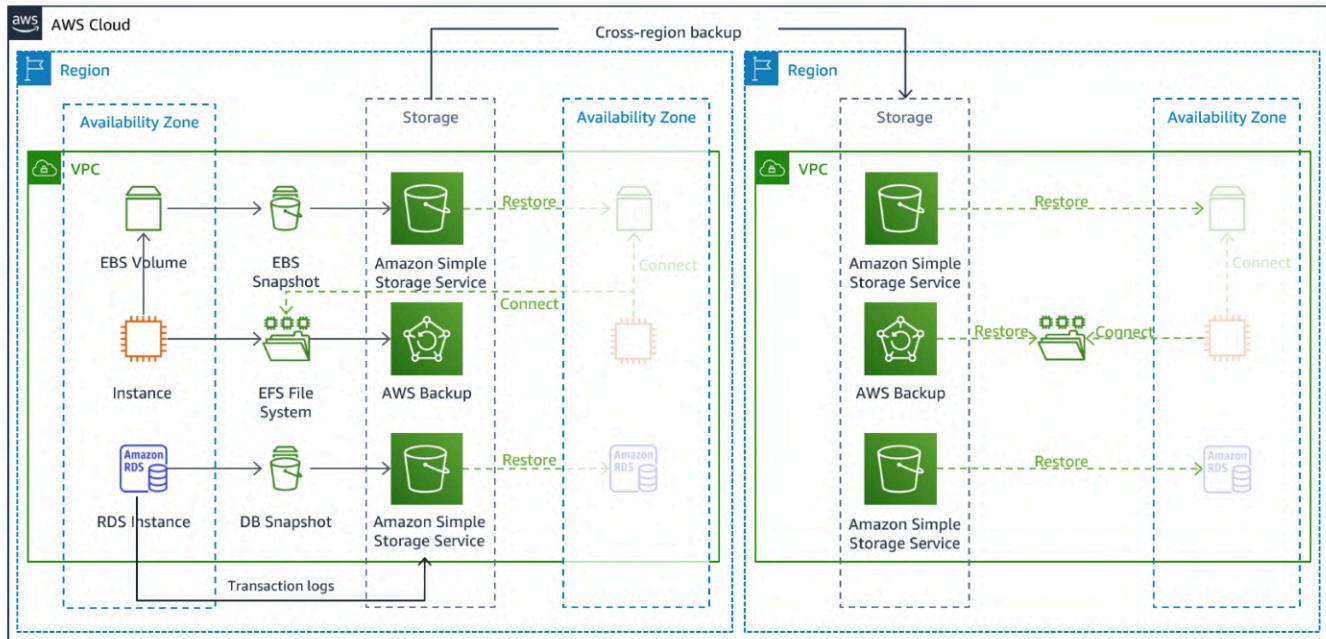


Figure 19: Backup and restore architecture

For more details on this strategy see [Disaster Recovery \(DR\) Architecture on AWS, Part II: Backup and Restore with Rapid Recovery](#).

Pilot light

With the *pilot light* approach, you replicate your data from your primary Region to your recovery Region. Core resources used for the workload infrastructure are deployed in the recovery Region, however additional resources and any dependencies are still needed to make this a functional stack. For example, in Figure 20, no compute instances are deployed.

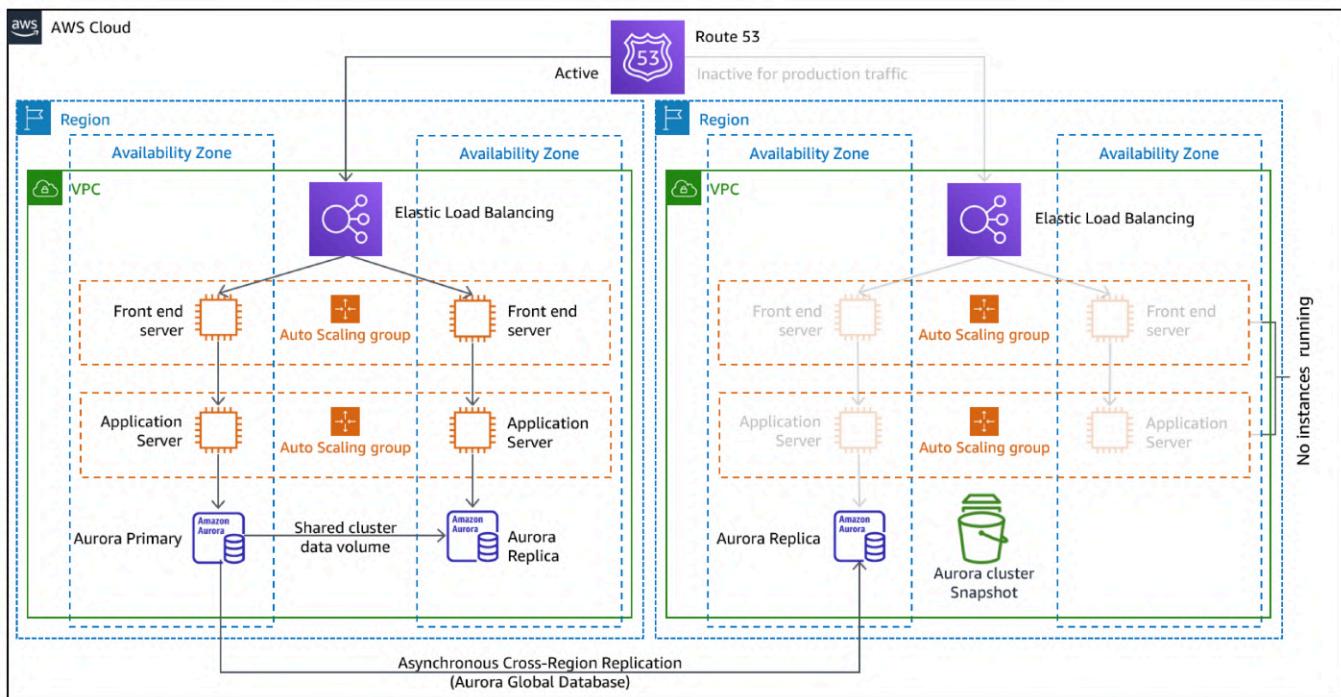


Figure 20: Pilot light architecture

For more details on this strategy, see [Disaster Recovery \(DR\) Architecture on AWS, Part III: Pilot Light and Warm Standby](#).

Warm standby

The *warm standby* approach involves ensuring that there is a scaled down, but fully functional, copy of your production environment in another Region. This approach extends the pilot light concept and decreases the time to recovery because your workload is always-on in another Region. If the recovery Region is deployed at full capacity, then this is known as *hot standby*.

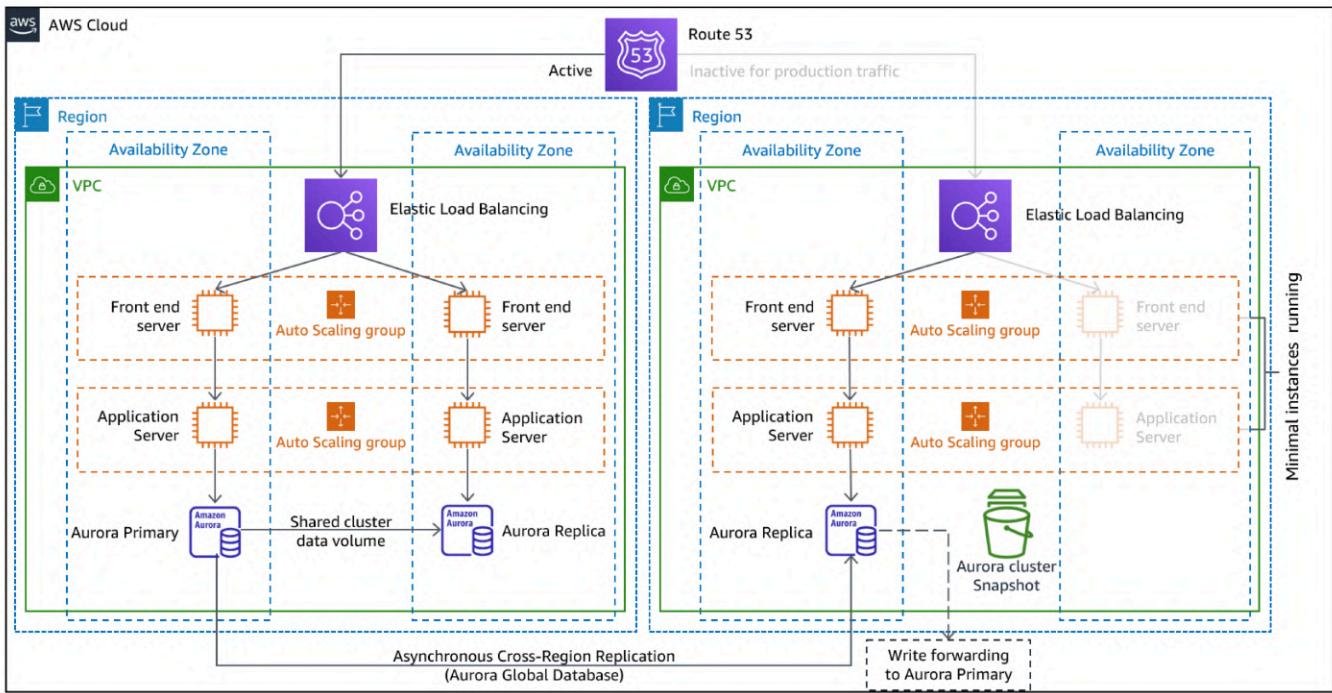


Figure 21: Warm standby architecture

Using warm standby or pilot light requires scaling up resources in the recovery Region. To verify capacity is available when needed, consider the use for [capacity reservations](#) for EC2 instances. If using AWS Lambda, then [provisioned concurrency](#) can provide runtime environments so that they are prepared to respond immediately to your function's invocations.

For more details on this strategy, see [Disaster Recovery \(DR\) Architecture on AWS, Part III: Pilot Light and Warm Standby](#).

Multi-site active/active

You can run your workload simultaneously in multiple Regions as part of a *multi-site active/active* strategy. Multi-site active/active serves traffic from all regions to which it is deployed. Customers may select this strategy for reasons other than DR. It can be used to increase availability, or when deploying a workload to a global audience (to put the endpoint closer to users and/or to deploy stacks localized to the audience in that region). As a DR strategy, if the workload cannot be supported in one of the AWS Regions to which it is deployed, then that Region is evacuated, and the remaining Regions are used to maintain availability. Multi-site active/active is the most operationally complex of the DR strategies, and should only be selected when business requirements necessitate it.

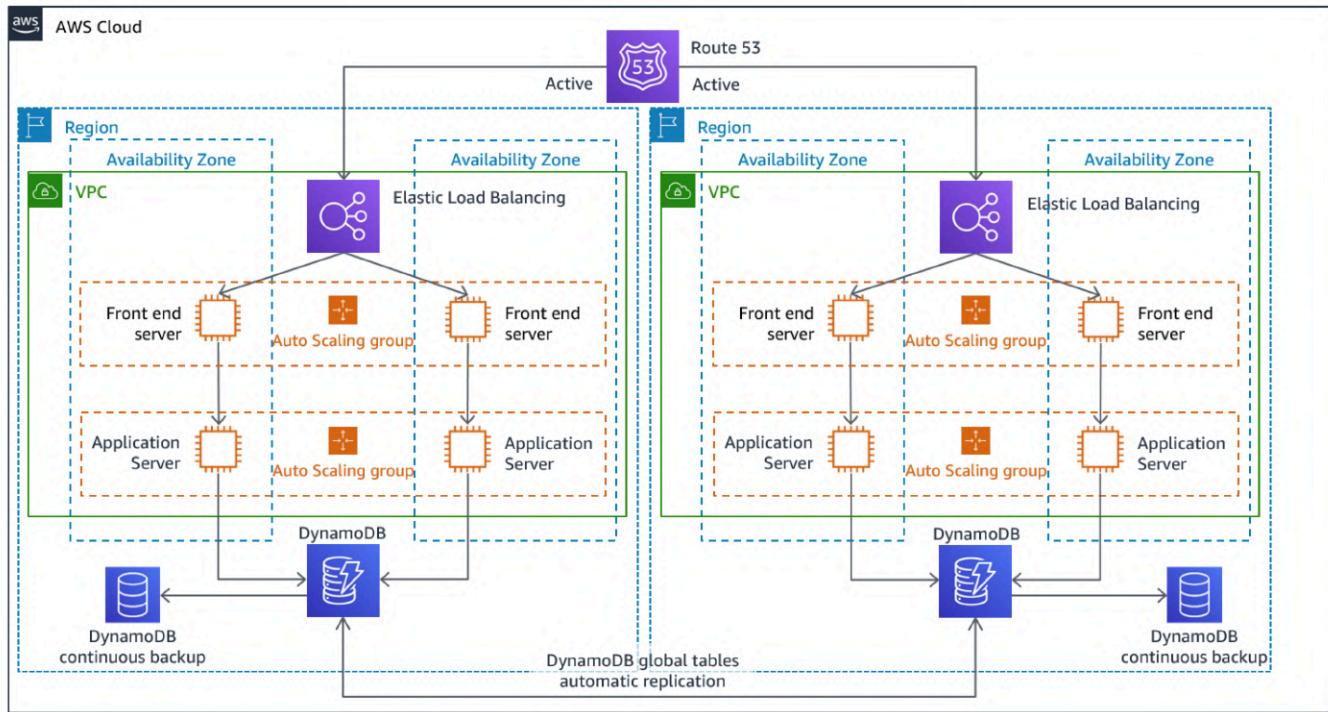


Figure 22: Multi-site active/active architecture

For more details on this strategy, see [Disaster Recovery \(DR\) Architecture on AWS, Part IV: Multi-site Active/Active](#).

AWS Elastic Disaster Recovery

If you are considering the pilot light or warm standby strategy for disaster recovery, AWS Elastic Disaster Recovery could provide an alternative approach with improved benefits. Elastic Disaster Recovery can offer an RPO and RTO target similar to warm standby, but maintain the low-cost approach of pilot light. Elastic Disaster Recovery replicates your data from your primary region to your recovery Region, using continual data protection to achieve an RPO measured in seconds and an RTO that can be measured in minutes. Only the resources required to replicate the data are deployed in the recovery region, which keeps costs down, similar to the pilot light strategy. When using Elastic Disaster Recovery, the service coordinates and orchestrates the recovery of compute resources when initiated as part of failover or drill.

AWS Elastic Disaster Recovery (AWS DRS) general architecture

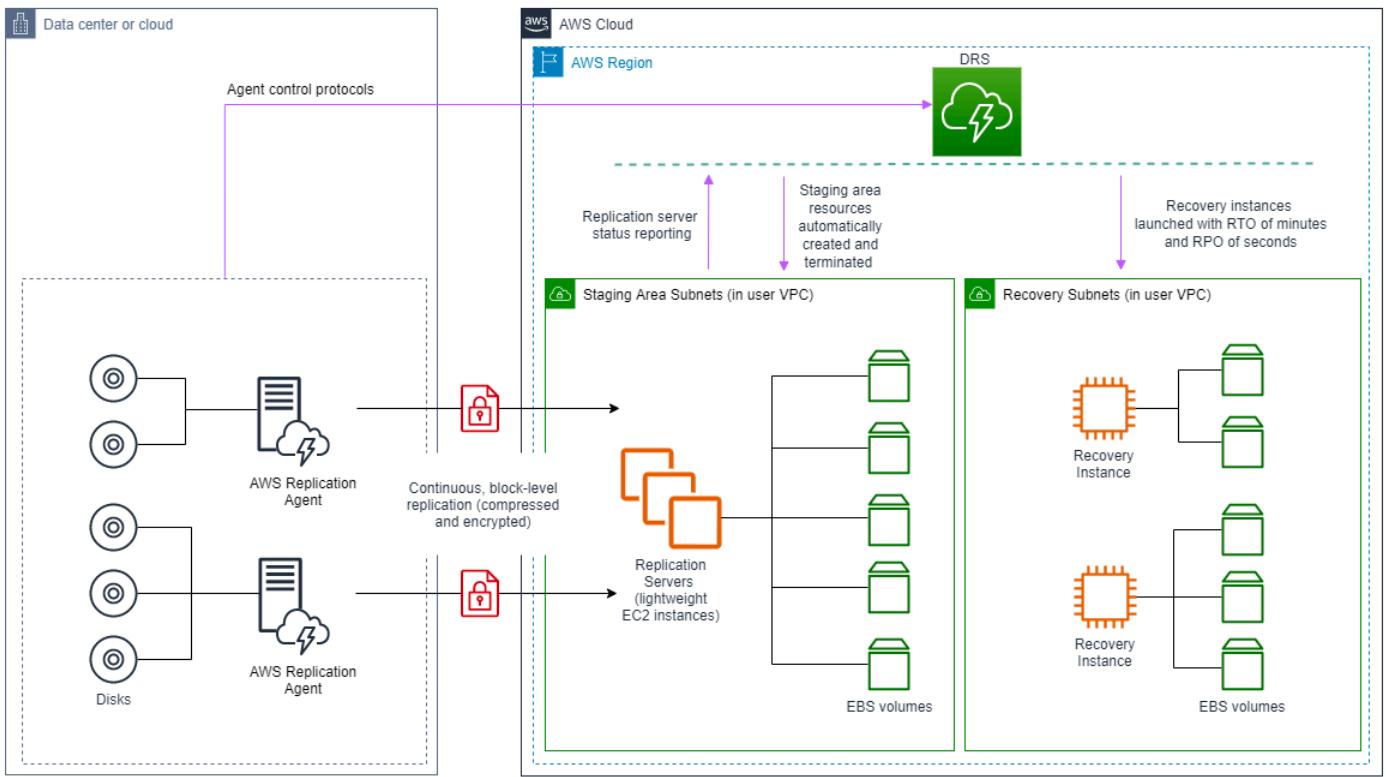


Figure 23: AWS Elastic Disaster Recovery architecture

Additional practices for protecting data

With all strategies, you must also mitigate against a data disaster. Continuous data replication protects you against some types of disaster, but it may not protect you against data corruption or destruction unless your strategy also includes versioning of stored data or options for point-in-time recovery. You must also back up the replicated data in the recovery site to create point-in-time backups in addition to the replicas.

Using multiple Availability Zones (AZs) within a single AWS Region

When using multiple AZs within a single Region, your DR implementation uses multiple elements of the above strategies. First you must create a high-availability (HA) architecture, using multiple AZs as shown in Figure 23. This architecture makes use of a multi-site active/active approach, as the [Amazon EC2 instances](#) and the [Elastic Load Balancer](#) have resources deployed in multiple AZs, actively handing requests. The architecture also demonstrates hot

standby, where if the primary [Amazon RDS](#) instance fails (or the AZ itself fails), then the standby instance is promoted to primary.

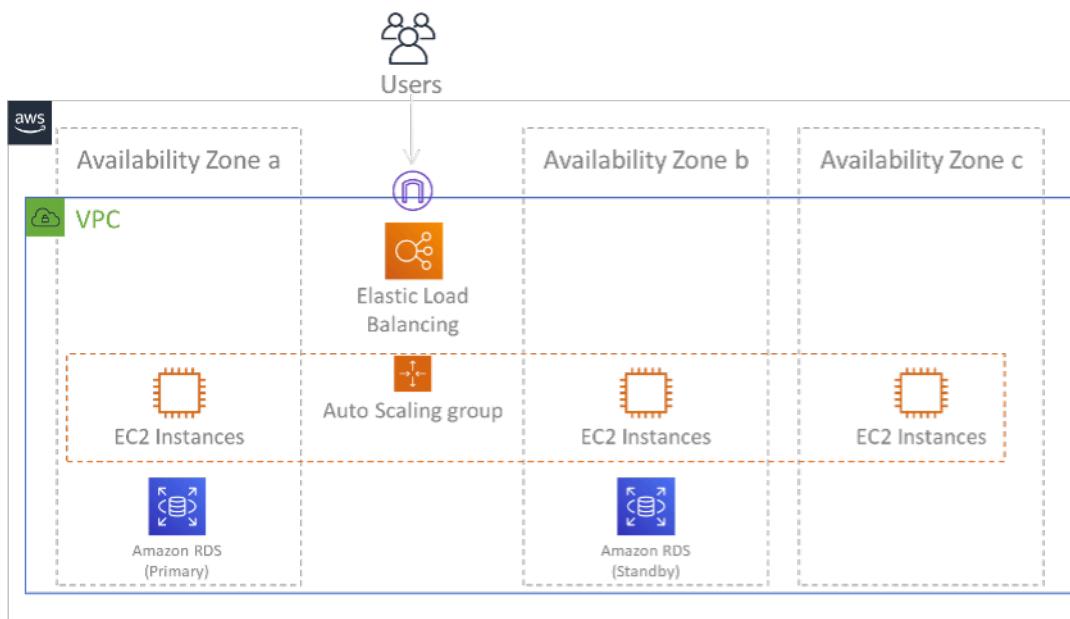


Figure 24: Multi-AZ architecture

In addition to this HA architecture, you need to add backups of all data required to run your workload. This is especially important for data that is constrained to a single zone such as [Amazon EBS volumes](#) or [Amazon Redshift clusters](#). If an AZ fails, you will need to restore this data to another AZ. Where possible, you should also copy data backups to another AWS Region as an additional layer of protection.

An less common alternative approach to single Region, multi-AZ DR is illustrated in the blog post, [Building highly resilient applications using Amazon Route 53 Application Recovery Controller, Part 1: Single-Region stack](#). Here, the strategy is to maintain as much isolation between the AZs as possible, like how Regions operate. Using this alternative strategy, you can choose an active/active or active/passive approach.

Note

Some workloads have regulatory data residency requirements. If this applies to your workload in a locality that currently has only one AWS Region, then multi-Region will not suit your business needs. Multi-AZ strategies provide good protection against most disasters.

3. Assess the resources of your workload, and what their configuration will be in the recovery Region prior to failover (during normal operation).

For infrastructure and AWS resources use infrastructure as code such as [AWS CloudFormation](#) or third-party tools like Hashicorp Terraform. To deploy across multiple accounts and Regions with a single operation you can use [AWS CloudFormation StackSets](#). For Multi-site active/active and Hot Standby strategies, the deployed infrastructure in your recovery Region has the same resources as your primary Region. For Pilot Light and Warm Standby strategies, the deployed infrastructure will require additional actions to become production ready. Using CloudFormation [parameters](#) and [conditional logic](#), you can control whether a deployed stack is active or standby with [a single template](#). When using Elastic Disaster Recovery, the service will replicate and orchestrate the restoration of application configurations and compute resources.

All DR strategies require that data sources are backed up within the AWS Region, and then those backups are copied to the recovery Region. [AWS Backup](#) provides a centralized view where you can configure, schedule, and monitor backups for these resources. For Pilot Light, Warm Standby, and Multi-site active/active, you should also replicate data from the primary Region to data resources in the recovery Region, such as [Amazon Relational Database Service \(Amazon RDS\)](#) DB instances or [Amazon DynamoDB](#) tables. These data resources are therefore live and ready to serve requests in the recovery Region.

To learn more about how AWS services operate across Regions, see this blog series on [Creating a Multi-Region Application with AWS Services](#).

4. Determine and implement how you will make your recovery Region ready for failover when needed (during a disaster event).

For multi-site active/active, failover means evacuating a Region, and relying on the remaining active Regions. In general, those Regions are ready to accept traffic. For Pilot Light and Warm Standby strategies, your recovery actions will need to deploy the missing resources, such as the EC2 instances in Figure 20, plus any other missing resources.

For all of the above strategies you may need to promote read-only instances of databases to become the primary read/write instance.

For backup and restore, restoring data from backup creates resources for that data such as EBS volumes, RDS DB instances, and DynamoDB tables. You also need to restore the infrastructure and deploy code. You can use AWS Backup to restore data in the recovery Region. See [REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from](#)

[sources](#) for more details. Rebuilding the infrastructure includes creating resources like EC2 instances in addition to the [Amazon Virtual Private Cloud \(Amazon VPC\)](#), subnets, and security groups needed. You can automate much of the restoration process. To learn how, see [this blog post](#).

5. Determine and implement how you will reroute traffic to failover when needed (during a disaster event).

This failover operation can be initiated either automatically or manually. Automatically initiated failover based on health checks or alarms should be used with caution since an unnecessary failover (false alarm) incurs costs such as non-availability and data loss. Manually initiated failover is therefore often used. In this case, you should still automate the steps for failover, so that the manual initiation is like the push of a button.

There are several traffic management options to consider when using AWS services. One option is to use [Amazon Route 53](#). Using Amazon Route 53, you can associate multiple IP endpoints in one or more AWS Regions with a Route 53 domain name. To implement manually initiated failover you can use [Amazon Route 53 Application Recovery Controller](#), which provides a highly available data plane API to reroute traffic to the recovery Region. When implementing failover, use data plane operations and avoid control plane ones as described in [REL11-BP04 Rely on the data plane and not the control plane during recovery](#).

To learn more about this and other options see [this section of the Disaster Recovery Whitepaper](#).

6. Design a plan for how your workload will fail back.

Failback is when you return workload operation to the primary Region, after a disaster event has abated. Provisioning infrastructure and code to the primary Region generally follows the same steps as were initially used, relying on infrastructure as code and code deployment pipelines. The challenge with failback is restoring data stores, and ensuring their consistency with the recovery Region in operation.

In the failed over state, the databases in the recovery Region are live and have the up-to-date data. The goal then is to re-synchronize from the recovery Region to the primary Region, ensuring it is up-to-date.

Some AWS services will do this automatically. If using [Amazon DynamoDB global tables](#), even if the table in the primary Region had become not available, when it comes back online, DynamoDB resumes propagating any pending writes. If using [Amazon Aurora Global Database](#) and using [managed planned failover](#), then Aurora global database's existing replication topology

is maintained. Therefore, the former read/write instance in the primary Region will become a replica and receive updates from the recovery Region.

In cases where this is not automatic, you will need to re-establish the database in the primary Region as a replica of the database in the recovery Region. In many cases this will involve deleting the old primary database, and creating new replicas.

After a failover, if you can continue running in your recovery Region, consider making this the new primary Region. You would still do all the above steps to make the former primary Region into a recovery Region. Some organizations do a scheduled rotation, swapping their primary and recovery Regions periodically (for example every three months).

All of the steps required to fail over and fail back should be maintained in a playbook that is available to all members of the team, and is periodically reviewed.

When using Elastic Disaster Recovery, the service will assist in orchestrating and automating the failback process. For more details, see [Performing a failback](#).

Level of effort for the Implementation Plan: High

Resources

Related best practices:

- [the section called “REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources”](#)
- [the section called “REL11-BP04 Rely on the data plane and not the control plane during recovery”](#)
- [the section called “REL13-BP01 Define recovery objectives for downtime and data loss”](#)

Related documents:

- [AWS Architecture Blog: Disaster Recovery Series](#)
- [Disaster Recovery of Workloads on AWS: Recovery in the Cloud \(AWS Whitepaper\)](#)
- [Disaster recovery options in the cloud](#)
- [Build a serverless multi-region, active-active backend solution in an hour](#)
- [Multi-region serverless backend — reloaded](#)

- [RDS: Replicating a Read Replica Across Regions](#)
- [Route 53: Configuring DNS Failover](#)
- [S3: Cross-Region Replication](#)
- [What Is AWS Backup?](#)
- [What is Route 53 Application Recovery Controller?](#)
- [AWS Elastic Disaster Recovery](#)
- [HashiCorp Terraform: Get Started - AWS](#)
- [APN Partner: partners that can help with disaster recovery](#)
- [AWS Marketplace: products that can be used for disaster recovery](#)

Related videos:

- [Disaster Recovery of Workloads on AWS](#)
- [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications \(ARC209-R2\)](#)
- [Get Started with AWS Elastic Disaster Recovery | Amazon Web Services](#)

Related examples:

- [Well-Architected Lab - Disaster Recovery](#) - Series of workshops illustrating DR strategies

REL13-BP03 Test disaster recovery implementation to validate the implementation

Regularly test failover to your recovery site to verify that it operates properly and that RTO and RPO are met.

Common anti-patterns:

- Never exercise failovers in production.

Benefits of establishing this best practice: Regularly testing your disaster recovery plan verifies that it will work when it needs to, and that your team knows how to perform the strategy.

Level of risk exposed if this best practice is not established: High

Implementation guidance

A pattern to avoid is developing recovery paths that are rarely exercised. For example, you might have a secondary data store that is used for read-only queries. When you write to a data store and the primary fails, you might want to fail over to the secondary data store. If you don't frequently test this failover, you might find that your assumptions about the capabilities of the secondary data store are incorrect. The capacity of the secondary, which might have been sufficient when you last tested, might be no longer be able to tolerate the load under this scenario. Our experience has shown that the only error recovery that works is the path you test frequently. This is why having a small number of recovery paths is best. You can establish recovery patterns and regularly test them. If you have a complex or critical recovery path, you still need to regularly exercise that failure in production to convince yourself that the recovery path works. In the example we just discussed, you should fail over to the standby regularly, regardless of need.

Implementation steps

1. Engineer your workloads for recovery. Regularly test your recovery paths. Recovery-oriented computing identifies the characteristics in systems that enhance recovery: isolation and redundancy, system-wide ability to roll back changes, ability to monitor and determine health, ability to provide diagnostics, automated recovery, modular design, and ability to restart. Exercise the recovery path to verify that you can accomplish the recovery in the specified time to the specified state. Use your runbooks during this recovery to document problems and find solutions for them before the next test.
2. For Amazon EC2-based workloads, use [AWS Elastic Disaster Recovery](#) to implement and launch drill instances for your DR strategy. AWS Elastic Disaster Recovery provides the ability to efficiently run drills, which helps you prepare for a failover event. You can also frequently launch of your instances using Elastic Disaster Recovery for test and drill purposes without redirecting the traffic.

Resources

Related documents:

- [APN Partner: partners that can help with disaster recovery](#)
- [AWS Architecture Blog: Disaster Recovery Series](#)
- [AWS Marketplace: products that can be used for disaster recovery](#)
- [AWS Elastic Disaster Recovery](#)

- [Disaster Recovery of Workloads on AWS: Recovery in the Cloud \(AWS Whitepaper\)](#)
- [AWS Elastic Disaster Recovery Preparing for Failover](#)
- [The Berkeley/Stanford recovery-oriented computing project](#)
- [What is AWS Fault Injection Simulator?](#)

Related videos:

- [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications](#)
- [AWS re:Invent 2019: Backup-and-restore and disaster-recovery solutions with AWS](#)

Related examples:

- [Well-Architected Lab - Testing for Resiliency](#)

REL13-BP04 Manage configuration drift at the DR site or Region

Ensure that the infrastructure, data, and configuration are as needed at the DR site or Region. For example, check that AMIs and service quotas are up to date.

AWS Config continuously monitors and records your AWS resource configurations. It can detect drift and invoke [AWS Systems Manager Automation](#) to fix it and raise alarms. AWS CloudFormation can additionally detect drift in stacks you have deployed.

Common anti-patterns:

- Failing to make updates in your recovery locations, when you make configuration or infrastructure changes in your primary locations.
- Not considering potential limitations (like service differences) in your primary and recovery locations.

Benefits of establishing this best practice: Ensuring that your DR environment is consistent with your existing environment guarantees complete recovery.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Ensure that your delivery pipelines deliver to both your primary and backup sites. Delivery pipelines for deploying applications into production must distribute to all the specified disaster recovery strategy locations, including dev and test environments.
- Permit AWS Config to track potential drift locations. Use AWS Config rules to create systems that enforce your disaster recovery strategies and generate alerts when they detect drift.
 - [Remediating Noncompliant AWS Resources by AWS Config Rules](#)
 - [AWS Systems Manager Automation](#)
- Use AWS CloudFormation to deploy your infrastructure. AWS CloudFormation can detect drift between what your CloudFormation templates specify and what is actually deployed.
 - [AWS CloudFormation: Detect Drift on an Entire CloudFormation Stack](#)

Resources

Related documents:

- [APN Partner: partners that can help with disaster recovery](#)
- [AWS Architecture Blog: Disaster Recovery Series](#)
- [AWS CloudFormation: Detect Drift on an Entire CloudFormation Stack](#)
- [AWS Marketplace: products that can be used for disaster recovery](#)
- [AWS Systems Manager Automation](#)
- [Disaster Recovery of Workloads on AWS: Recovery in the Cloud \(AWS Whitepaper\)](#)
- [How do I implement an Infrastructure Configuration Management solution on AWS?](#)
- [Remediating Noncompliant AWS Resources by AWS Config Rules](#)

Related videos:

- [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications \(ARC209-R2\)](#)

REL13-BP05 Automate recovery

Use AWS or third-party tools to automate system recovery and route traffic to the DR site or Region.

Based on configured health checks, AWS services, such as Elastic Load Balancing and AWS Auto Scaling, can distribute load to healthy Availability Zones while services, such as Amazon Route 53 and AWS Global Accelerator, can route load to healthy AWS Regions. Amazon Route 53 Application Recovery Controller helps you manage and coordinate failover using readiness check and routing control features. These features continually monitor your application's ability to recover from failures, so you can control application recovery across multiple AWS Regions, Availability Zones, and on premises.

For workloads on existing physical or virtual data centers or private clouds, [AWS Elastic Disaster Recovery](#) allows organizations to set up an automated disaster recovery strategy in AWS. Elastic Disaster Recovery also supports cross-Region and cross-Availability Zone disaster recovery in AWS.

Common anti-patterns:

- Implementing identical automated failover and fallback can cause flapping when a failure occurs.

Benefits of establishing this best practice: Automated recovery reduces your recovery time by eliminating the opportunity for manual errors.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

- Automate recovery paths. For short recovery times, follow your [disaster recovery plan](#) to get your IT systems back online quickly in the case of a disruption.
 - Use Elastic Disaster Recovery for automated Failover and Failback. Elastic Disaster Recovery continuously replicates your machines (including operating system, system state configuration, databases, applications, and files) into a low-cost staging area in your target AWS account and preferred Region. In the case of a disaster, after choosing to recover using Elastic Disaster Recovery, Elastic Disaster Recovery automates the conversion of your replicated servers into fully provisioned workloads in your recovery Region on AWS.
 - [Using Elastic Disaster Recovery for Failover and Failback](#)
 - [AWS Elastic Disaster Recovery resources](#)

Resources

Related documents:

- [APN Partner: partners that can help with disaster recovery](#)

- [AWS Architecture Blog: Disaster Recovery Series](#)
- [AWS Marketplace: products that can be used for disaster recovery](#)
- [AWS Systems Manager Automation](#)
- [AWS Elastic Disaster Recovery](#)
- [Disaster Recovery of Workloads on AWS: Recovery in the Cloud \(AWS Whitepaper\)](#)

Related videos:

- [AWS re:Invent 2018: Architecture Patterns for Multi-Region Active-Active Applications \(ARC209-R2\)](#)

Performance efficiency

The Performance Efficiency pillar includes the ability to use computing resources efficiently to meet system requirements, and to maintain that efficiency as demand changes and technologies evolve. You can find prescriptive guidance on implementation in the [Performance Efficiency Pillar whitepaper](#).

Best practice areas

- [Architecture selection](#)
- [Compute and hardware](#)
- [Data management](#)
- [Networking and content delivery](#)
- [Process and culture](#)

Architecture selection

Questions

- [PERF 1. How do you select appropriate cloud resources and architecture for your workload?](#)

PERF 1. How do you select appropriate cloud resources and architecture for your workload?

The optimal solution for a particular workload varies, and solutions often combine multiple approaches. Well-Architected workloads use multiple solutions and allow different features to improve performance.

Best practices

- [PERF01-BP01 Learn about and understand available cloud services and features](#)
- [PERF01-BP02 Use guidance from your cloud provider or an appropriate partner to learn about architecture patterns and best practices](#)
- [PERF01-BP03 Factor cost into architectural decisions](#)
- [PERF01-BP04 Evaluate how trade-offs impact customers and architecture efficiency](#)
- [PERF01-BP05 Use policies and reference architectures](#)
- [PERF01-BP06 Use benchmarking to drive architectural decisions](#)
- [PERF01-BP07 Use a data-driven approach for architectural choices](#)

PERF01-BP01 Learn about and understand available cloud services and features

Continually learn about and discover available services and configurations that help you make better architectural decisions and improve performance efficiency in your workload architecture.

Common anti-patterns:

- You use the cloud as a collocated data center.
- You do not modernize your application after migration to the cloud.
- You only use one storage type for all things that need to be persisted.
- You use instance types that are closest matched to your current standards, but are larger where needed.
- You deploy and manage technologies that are available as managed services.

Benefits of establishing this best practice: By considering new services and configurations, you may be able to greatly improve performance, reduce cost, and optimize the effort required to maintain your workload. It can also help you accelerate the time-to-value for cloud-enabled products.

Level of risk exposed if this best practice is not established: High

Implementation guidance

AWS continually releases new services and features that can improve performance and reduce the cost of cloud workloads. Staying up-to-date with these new services and features is crucial for maintaining performance efficacy in the cloud. Modernizing your workload architecture also helps you accelerate productivity, drive innovation, and unlock more growth opportunities.

Implementation steps

- Inventory your workload software and architecture for related services. Decide which category of products to learn more about.
- Explore AWS offerings to identify and learn about the relevant services and configuration options that can help you improve performance and reduce cost and operational complexity.
 - [What's New with AWS?](#)
 - [AWS Blog](#)
 - [AWS Skill Builder](#)
 - [AWS Events and Webinars](#)
 - [AWS Training and Certifications](#)
 - [AWS Youtube Channel](#)
 - [AWS Workshops](#)
 - [AWS Communities](#)
- Use sandbox (non-production) environments to learn and experiment with new services without incurring extra cost.

Resources

Related documents:

- [AWS Architecture Center](#)
- [AWS Partner Network](#)
- [AWS Solutions Library](#)
- [AWS Knowledge Center](#)
- [Build modern applications on AWS](#)

Related videos:

- [This is my Architecture](#)

Related examples:

- [AWS Samples](#)
- [AWS SDK Examples](#)

PERF01-BP02 Use guidance from your cloud provider or an appropriate partner to learn about architecture patterns and best practices

Use cloud company resources such as documentation, solutions architects, professional services, or appropriate partners to guide your architectural decisions. These resources help you review and improve your architecture for optimal performance.

Common anti-patterns:

- You use AWS as a common cloud provider.
- You use AWS services in a manner that they were not designed for.
- You follow all guidance without considering your business context.

Benefits of establishing this best practice: Using guidance from a cloud provider or an appropriate partner can help you to make the right architectural choices for your workload and give you confidence in your decisions.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

AWS offers a wide range of guidance, documentation, and resources that can help you build and manage efficient cloud workloads. AWS documentation provides code samples, tutorials, and detailed service explanations. In addition to documentation, AWS provides training and certification programs, solutions architects, and professional services that can help customers explore different aspects of cloud services and implement efficient cloud architecture on AWS.

Leverage these resources to gain insights into valuable knowledge and best practices, save time, and achieve better outcomes in the AWS Cloud.

Implementation steps

- Review AWS documentation and guidance and follow the best practices. These resources can help you effectively choose and configure services and achieve better performance.
 - [AWS documentation](#) (like user guides and whitepapers)
 - [AWS Blog](#)
 - [AWS Training and Certifications](#)
 - [AWS Youtube Channel](#)
- Join AWS partner events (like AWS Global Summits, AWS re:Invent, user groups, and workshops) to learn from AWS experts about best practices for using AWS services.
 - [AWS Events and Webinars](#)
 - [AWS Workshops](#)
 - [AWS Communities](#)
- Reach out to AWS for assistance when you need additional guidance or product information. AWS Solutions Architects and [AWS Professional Services](#) provide guidance for solution implementation. [AWS Partners](#) provide AWS expertise to help you unlock agility and innovation for your business.
- Use [AWS Support](#) if you need technical support to use a service effectively. [Our Support plans](#) are designed to give you the right mix of tools and access to expertise so that you can be successful with AWS while optimizing performance, managing risk, and keeping costs under control.

Resources

Related documents:

- [AWS Architecture Center](#)
- [AWS Solutions Library](#)
- [AWS Knowledge Center](#)
- [AWS Enterprise Support](#)

Related videos:

- [This is my Architecture](#)

Related examples:

- [AWS Samples](#)
- [AWS SDK Examples](#)

PERF01-BP03 Factor cost into architectural decisions

Factor cost into your architectural decisions to improve resource utilization and performance efficiency of your cloud workload. When you are aware of the cost implications of your cloud workload, you are more likely to leverage efficient resources and reduce wasteful practices.

Common anti-patterns:

- You only use one family of instances.
- You do not evaluate licensed solutions against open-source solutions.
- You do not define storage lifecycle policies.
- You do not review new services and features of the AWS Cloud.
- You only use block storage.

Benefits of establishing this best practice: Factoring cost into your decision making allows you to use more efficient resources and explore other investments.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Optimizing workloads for cost can improve resource utilization and avoid waste in a cloud workload. Factoring cost into architectural decisions usually includes right-sizing workload components and enabling elasticity, which results in improved cloud workload performance efficiency.

Implementation steps

- Establish cost objectives like budget limits for your cloud workload.
- Identify the key components (like instances and storage) that drive cost of your workload. You can use [AWS Pricing Calculator](#) and [AWS Cost Explorer](#) to identify key cost drivers in your workload.

- Use [Well-Architected cost optimization best practices](#) to optimize these key components for cost.
- Continually monitor and analyze cost to identify cost optimization opportunities in your workload.
 - Use [AWS Budgets](#) to get alerts for unacceptable costs.
 - Use [AWS Compute Optimizer](#) or [AWS Trusted Advisor](#) to get cost optimization recommendations.
 - Use [AWS Cost Anomaly Detection](#) to get automated cost anomaly detection and root cause analysis.

Resources

Related documents:

- [A Detailed Overview of the Cost Intelligence Dashboard](#)
- [AWS Architecture Center](#)
- [AWS Partner Network](#)
- [AWS Solutions Library](#)
- [AWS Knowledge Center](#)

Related videos:

- [This is my Architecture](#)
- [Optimize performance and cost for your AWS compute](#)

Related examples:

- [AWS Samples](#)
- [AWS SDK Examples](#)
- [Rightsizing with Compute Optimizer and Memory utilization enabled](#)
- [AWS Compute Optimizer Demo code](#)

PERF01-BP04 Evaluate how trade-offs impact customers and architecture efficiency

When evaluating performance-related improvements, determine which choices impact your customers and workload efficiency. For example, if using a key-value data store increases system

performance, it is important to evaluate how the eventually consistent nature of this change will impact customers.

Common anti-patterns:

- You assume that all performance gains should be implemented, even if there are tradeoffs for implementation.
- You only evaluate changes to workloads when a performance issue has reached a critical point.

Benefits of establishing this best practice: When you are evaluating potential performance-related improvements, you must decide if the tradeoffs for the changes are acceptable with the workload requirements. In some cases, you may have to implement additional controls to compensate for the tradeoffs.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Identify critical areas in your architecture in terms of performance and customer impact. Determine how you can make improvements, what trade-offs those improvements bring, and how they impact the system and the user experience. For example, implementing caching data can help dramatically improve performance but requires a clear strategy for how and when to update or invalidate cached data to prevent incorrect system behavior.

Implementation steps

- Understand your workload requirements and SLAs.
- Clearly define evaluation factors. Factors may relate to cost, reliability, security, and performance of your workload.
- Select architecture and services that can address your requirements.
- Conduct experimentation and proof of concepts (POCs) to evaluate trade-off factors and impact on customers and architecture efficiency. Usually, highly-available, performant, and secure workloads consume more cloud resources while providing better customer experience.

Resources

Related documents:

- [Amazon Builders' Library](#)

- [Amazon QuickSight KPIs](#)
- [Amazon CloudWatch RUM](#)
- [X-Ray Documentation](#)
- [Understand resiliency patterns and trade-offs to architect efficiently in the cloud](#)

Related videos:

- [Build a Monitoring Plan](#)
- [Optimize applications through Amazon CloudWatch RUM](#)
- [Demo of Amazon CloudWatch Synthetics](#)

Related examples:

- [Measure page load time with Amazon CloudWatch Synthetics](#)
- [Amazon CloudWatch RUM Web Client](#)

PERF01-BP05 Use policies and reference architectures

Use internal policies and existing reference architectures when selecting services and configurations to be more efficient when designing and implementing your workload.

Common anti-patterns:

- You allow a wide variety of technology that may impact the management overhead of your company.

Benefits of establishing this best practice: Establishing a policy for architecture, technology, and vendor choices allows decisions to be made quickly.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Having internal policies in selecting resources and architecture provides standards and guidelines to follow when making architectural choices. Those guidelines streamline the decision-making process when choosing the right cloud service and can help improve performance efficiency. Deploy your workload using policies or reference architectures. Integrate the services into your

cloud deployment, then use your performance tests to verify that you can continue to meet your performance requirements.

Implementation steps

- Clearly understand the requirements of your cloud workload.
- Review internal and external policies to identify the most relevant ones.
- Use the appropriate reference architectures provided by AWS or your industry best practices.
- Create a continuum consisting of policies, standards, reference architectures, and prescriptive guidelines for common situations. Doing so allows your teams to move faster. Tailor the assets for your vertical if applicable.
- Validate these policies and reference architectures for your workload in sandbox environments.
- Stay up-to-date with industry standards and AWS updates to make sure your policies and reference architectures help optimize your cloud workload.

Resources

Related documents:

- [AWS Architecture Center](#)
- [AWS Partner Network](#)
- [AWS Solutions Library](#)
- [AWS Knowledge Center](#)

Related videos:

- [This is my Architecture](#)

Related examples:

- [AWS Samples](#)
- [AWS SDK Examples](#)

PERF01-BP06 Use benchmarking to drive architectural decisions

Benchmark the performance of an existing workload to understand how it performs on the cloud and drive architectural decisions based on that data.

Common anti-patterns:

- You rely on common benchmarks that are not indicative of your workload's characteristics.
- You rely on customer feedback and perceptions as your only benchmark.

Benefits of establishing this best practice: Benchmarking your current implementation allows you to measure performance improvements.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Use benchmarking with synthetic tests to assess how your workload's components perform.

Benchmarking is generally quicker to set up than load testing and is used to evaluate the technology for a particular component. Benchmarking is often used at the start of a new project, when you lack a full solution to load test.

You can either build your own custom benchmark tests or use an industry standard test, such as [TPC-DS](#), to benchmark your workloads. Industry benchmarks are helpful when comparing environments. Custom benchmarks are useful for targeting specific types of operations that you expect to make in your architecture.

When benchmarking, it is important to pre-warm your test environment to get valid results. Run the same benchmark multiple times to verify that you've captured any variance over time.

Because benchmarks are generally faster to run than load tests, they can be used earlier in the deployment pipeline and provide faster feedback on performance deviations. When you evaluate a significant change in a component or service, a benchmark can be a quick way to see if you can justify the effort to make the change. Using benchmarking in conjunction with load testing is important because load testing informs you about how your workload performs in production.

Implementation steps

- Define the metrics (like CPU utilization, latency, or throughput) to evaluate your workload performance.

- Identify and set up a benchmarking tool that is suitable for your workload. You can use AWS services (like [Amazon CloudWatch](#)) or a third-party tool that is compatible with your workload.
- Perform your benchmark tests and monitor the metrics during the test.
- Analyze and document the benchmarking results to identify any bottlenecks and issues.
- Use the test results to make architectural decisions and adjust your workload. This may include changing services or adopting new features.
- Retest your workload after the adjustment.

Resources

Related documents:

- [AWS Architecture Center](#)
- [AWS Partner Network](#)
- [AWS Solutions Library](#)
- [AWS Knowledge Center](#)
- [Amazon CloudWatch RUM](#)
- [Amazon CloudWatch Synthetics](#)

Related videos:

- [This is my Architecture](#)
- [Optimize applications through Amazon CloudWatch RUM](#)
- [Demo of Amazon CloudWatch Synthetics](#)

Related examples:

- [AWS Samples](#)
- [AWS SDK Examples](#)
- [Distributed Load Tests](#)
- [Measure page load time with Amazon CloudWatch Synthetics](#)
- [Amazon CloudWatch RUM Web Client](#)

PERF01-BP07 Use a data-driven approach for architectural choices

Define a clear, data-driven approach for architectural choices to verify that the right cloud services and configurations are used to meet your specific business needs.

Common anti-patterns:

- You assume your current architecture is static and should not be updated over time.
- Your architectural choices are based upon guesses and assumptions.
- You introduce architecture changes over time without justification.

Benefits of establishing this best practice: By having a well-defined approach for making architectural choices, you use data to influence your workload design and make informed decisions over time.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Use internal experience and knowledge of the cloud or external resources such as published use cases or whitepapers to choose resources and services in your architecture. You should have a well-defined process that encourages experimentation and benchmarking with the services that could be used in your workload.

Backlogs for critical workloads should consist of not just user stories which deliver functionality relevant to business and users, but also technical stories which form an architecture runway for the workload. This runway is informed by new advancements in technology and new services and adopts them based on data and proper justification. This verifies that the architecture remains future-proof and does not stagnate.

Implementation steps

- Engage with key stakeholders to define workload requirements, including performance, availability, and cost considerations. Consider factors such as the number of users and usage pattern for your workload.
- Create an architecture runway or a technology backlog which is prioritized along with the functional backlog.
- Evaluate and assess different cloud services (for more detail, see [PERF01-BP01 Learn about and understand available cloud services and features](#)).

- Explore different architectural patterns, like microservices or serverless, that meet your performance requirements (for more detail, see [PERF01-BP02 Use guidance from your cloud provider or an appropriate partner to learn about architecture patterns and best practices](#)).
- Consult other teams, architecture diagrams, and resources, such as AWS Solution Architects, [AWS Architecture Center](#), and [AWS Partner Network](#), to help you choose the right architecture for your workload.
- Define performance metrics like throughput and response time that can help you evaluate the performance of your workload.
- Experiment and use defined metrics to validate the performance of the selected architecture.
- Continually monitor and make adjustments as needed to maintain the optimal performance of your architecture.
- Document your selected architecture and decisions as a reference for future updates and learnings.
- Continually review and update the architecture selection approach based on learnings, new technologies, and metrics that indicate a needed change or problem in the current approach.

Resources

Related documents:

- [AWS Solutions Library](#)
- [AWS Knowledge Center](#)

Related videos:

- [This is my Architecture](#)

Related examples:

- [AWS Samples](#)
- [AWS SDK Examples](#)

Compute and hardware

Questions

- [PERF 2. How do you select and use compute resources in your workload?](#)

PERF 2. How do you select and use compute resources in your workload?

The optimal compute choice for a particular workload can vary based on application design, usage patterns, and configuration settings. Architectures may use different compute choices for various components and allow different features to improve performance. Selecting the wrong compute choice for an architecture can lead to lower performance efficiency.

Best practices

- [PERF02-BP01 Select the best compute options for your workload](#)
- [PERF02-BP02 Understand the available compute configuration and features](#)
- [PERF02-BP03 Collect compute-related metrics](#)
- [PERF02-BP04 Configure and right-size compute resources](#)
- [PERF02-BP05 Scale your compute resources dynamically](#)
- [PERF02-BP06 Use optimized hardware-based compute accelerators](#)

PERF02-BP01 Select the best compute options for your workload

Selecting the most appropriate compute option for your workload allows you to improve performance, reduce unnecessary infrastructure costs, and lower the operational efforts required to maintain your workload.

Common anti-patterns:

- You use the same compute option that was used on premises.
- You lack awareness of the cloud compute options, features, and solutions, and how those solutions might improve your compute performance.
- You over-provision an existing compute option to meet scaling or performance requirements when an alternative compute option would align to your workload characteristics more precisely.

Benefits of establishing this best practice: By identifying the compute requirements and evaluating against the options available, you can make your workload more resource efficient.

Level of risk exposed if this best practice is not established: High

Implementation guidance

To optimize your cloud workloads for performance efficiency, it is important to select the most appropriate compute options for your use case and performance requirements. AWS provides a variety of compute options that cater to different workloads in the cloud. For instance, you can use [Amazon EC2](#) to launch and manage virtual servers, [AWS Lambda](#) to run code without having to provision or manage servers, [Amazon ECS](#) or [Amazon EKS](#) to run and manage containers, or [AWS Batch](#) to process large volumes of data in parallel. Based on your scale and compute needs, you should choose and configure the optimal compute solution for your situation. You can also consider using multiple types of compute solutions in a single workload, as each one has its own advantages and drawbacks.

The following steps guide you through selecting the right compute options to match your workload characteristics and performance requirements.

Implementation steps

1. Understand your workload compute requirements. Key requirements to consider include processing needs, traffic patterns, data access patterns, scaling needs, and latency requirements.
2. Learn about different compute options available for your workload on AWS (as outlined in [PERF01-BP01 Learn about and understand available cloud services and features](#)). Here are some key AWS compute options, their characteristics, and common use cases:

AWS service	Key characteristics	Common use cases
Amazon Elastic Compute Cloud (Amazon EC2)	Has dedicated option for hardware, license requirements, large selection of different instance families, processor types and compute accelerators	Lift and shift migrations, monolithic application, hybrid environments, enterprise applications
Amazon Elastic Container Service (Amazon ECS) , Amazon Elastic Kubernetes Service (Amazon EKS)	Easy deployment, consistent environments, scalable	Microservices, hybrid environments

AWS service	Key characteristics	Common use cases
AWS Lambda	Serverless compute service that runs code in response to events and automatically manages the underlying compute resources.	Microservices, event-driven applications
AWS Batch	Efficiently and dynamically provisions and scales Amazon Elastic Container Service (Amazon ECS) , Amazon Elastic Kubernetes Service (Amazon EKS) , and AWS Fargate compute resources, with an option to use On-Demand or Spot Instances based on your job requirements	HPC, train ML models
Amazon Lightsail	Preconfigured Linux and Windows application for running small workloads	Simple web applications, custom website

3. Evaluate cost (like hourly charge or data transfer) and management overhead (like patching and scaling) associated to each compute option.
4. Perform experiments and benchmarking in a non-production environment to identify which compute option can best address your workload requirements.
5. Once you have experimented and identified your new compute solution, plan your migration and validate your performance metrics.
6. Use AWS monitoring tools like [Amazon CloudWatch](#) and optimization services like [AWS Compute Optimizer](#) to continually optimize your compute resources based on real-world usage patterns.

Resources

Related documents:

- [Cloud Compute with AWS](#)
- [Amazon EC2 Instance Types](#)
- [Amazon EKS Containers: Amazon EKS Worker Nodes](#)
- [Amazon ECS Containers: Amazon ECS Container Instances](#)
- [Functions: Lambda Function Configuration](#)
- [Prescriptive Guidance for Containers](#)
- [Prescriptive Guidance for Serverless](#)

Related videos:

- [How to choose compute option for startups](#)
- [Optimize performance and cost for your AWS compute](#)
- [Amazon EC2 foundations](#)
- [Powering next-gen Amazon EC2: Deep dive into the Nitro system](#)
- [Deploy ML models for inference at high performance and low cost](#)
- [Better, faster, cheaper compute: Cost-optimizing Amazon EC2](#)

Related examples:

- [Migrating the Web application to containers](#)
- [Run a Serverless Hello World](#)

PERF02-BP02 Understand the available compute configuration and features

Understand the available configuration options and features for your compute service to help you provision the right amount of resources and improve performance efficiency.

Common anti-patterns:

- You do not evaluate compute options or available instance families against workload characteristics.
- You over-provision compute resources to meet peak-demand requirements.

Benefits of establishing this best practice: Be familiar with AWS compute features and configurations so that you can use a compute solution optimized to meet your workload characteristics and needs.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Each compute solution has unique configurations and features available to support different workload characteristics and requirements. Learn how these options complement your workload, and determine which configuration options are best for your application. Examples of these options include instance family, sizes, features (GPU, I/O), bursting, time-outs, function sizes, container instances, and concurrency. If your workload has been using the same compute option for more than four weeks and you anticipate that the characteristics will remain the same in the future, you can use [AWS Compute Optimizer](#) to find out if your current compute option is suitable for the workloads from CPU and memory perspective.

Implementation steps

1. Understand workload requirements (like CPU need, memory, and latency).
2. Review AWS documentation and best practices to learn about recommended configuration options that can help improve compute performance. Here are some key configuration options to consider:

Configuration option	Examples
Instance type	<ul style="list-style-type: none">• Compute-optimized instances are ideal for the workloads that require high vCPU to memory ratio.• Memory-optimized instances deliver large amounts of memory to support memory intensive workloads.• Storage-optimized instances are designed for workloads that require high, sequential read and write access (IOPS) to local storage.

Configuration option	Examples
Pricing model	<ul style="list-style-type: none"> • On-Demand Instances let you use the compute capacity by the hour or second with no long-term commitment. These instances are good for bursting above performance baseline needs. • Savings Plans offer significant savings over On-Demand Instances in exchange for a commitment to use a specific amount of compute power for a one or three-year period. • Spot Instances let you take advantage of unused instance capacity at a discount for your stateless, fault-tolerant workloads.
Auto Scaling	Use Auto Scaling configuration to match compute resources to traffic patterns.
Sizing	<ul style="list-style-type: none"> • Use Compute Optimizer to get a machine-learning powered recommendations on which compute configuration best matches your compute characteristics. • Use AWS Lambda Power Tuning to select the best configuration for your Lambda function.
Hardware-based compute accelerators	<ul style="list-style-type: none"> • Accelerated computing instances perform functions like graphics processing or data pattern matching more efficiently than CPU-based alternatives. • For machine learning workloads, take advantage of purpose-built hardware that is specific to your workload, such as AWS Trainium, AWS Inferentia, and Amazon EC2 DL1

Resources

Related documents:

- [Cloud Compute with AWS](#)
- [Amazon EC2 Instance Types](#)
- [Processor State Control for Your Amazon EC2 Instance](#)
- [Amazon EKS Containers: Amazon EKS Worker Nodes](#)
- [Amazon ECS Containers: Amazon ECS Container Instances](#)
- [Functions: Lambda Function Configuration](#)

Related videos:

- [Amazon EC2 foundations](#)
- [Powering next-gen Amazon EC2: Deep dive into the Nitro system](#)
- [Optimize performance and cost for your AWS compute](#)

Related examples:

- [Rightsizing with Compute Optimizer and Memory utilization enabled](#)
- [AWS Compute Optimizer Demo code](#)

PERF02-BP03 Collect compute-related metrics

Record and track compute-related metrics to better understand how your compute resources are performing and improve their performance and their utilization.

Common anti-patterns:

- You only use manual log file searching for metrics.
- You only use the default metrics recorded by your monitoring software.
- You only review metrics when there is an issue.

Benefits of establishing this best practice: Collecting performance-related metrics will help you align application performance with business requirements to ensure that you are meeting your

workload needs. It can also help you continually improve the resource performance and utilization in your workload.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Cloud workloads can generate large volumes of data such as metrics, logs, and events. In the AWS Cloud, collecting metrics is a crucial step to improve security, cost efficiency, performance, and sustainability. AWS provides a wide range of performance-related metrics using monitoring services such as [Amazon CloudWatch](#) to provide you with valuable insights. Metrics such as CPU utilization, memory utilization, disk I/O, and network inbound and outbound can provide insight into utilization levels or performance bottlenecks. Use these metrics as part of a data-driven approach to actively tune and optimize your workload's resources. In an ideal case, you should collect all metrics related to your compute resources in a single platform with retention policies implemented to support cost and operational goals.

Implementation steps

1. Identify which performance-related metrics are relevant to your workload. You should collect metrics around resource utilization and the way your cloud workload is operating (like response time and throughput).
 - a. [Amazon EC2 default metrics](#)
 - b. [Amazon ECS default metrics](#)
 - c. [Amazon EKS default metrics](#)
 - d. [Lambda default metrics](#)
 - e. [Amazon EC2 memory and disk metrics](#)
2. Choose and set up the right logging and monitoring solution for your workload.
 - a. [AWS native Observability](#)
 - b. [AWS Distro for OpenTelemetry](#)
 - c. [Amazon Managed Service for Prometheus](#)
3. Define the required filter and aggregation for the metrics based on your workload requirements.
 - a. [Quantify custom application metrics with Amazon CloudWatch Logs and metric filters](#)
 - b. [Collect custom metrics with Amazon CloudWatch strategic tagging](#)
4. Configure data retention policies for your metrics to match your security and operational goals.
 - a. [Default data retention for CloudWatch metrics](#)

- b. [Default data retention for CloudWatch Logs](#)
- 5. If required, create alarms and notifications for your metrics to help you proactively respond to performance-related issues.
 - a. [Create alarms for custom metrics using Amazon CloudWatch anomaly detection](#)
 - b. [Create metrics and alarms for specific web pages with Amazon CloudWatch RUM](#)
- 6. Use automation to deploy your metric and log aggregation agents.
 - a. [AWS Systems Manager automation](#)
 - b. [OpenTelemetry Collector](#)

Resources

Related documents:

- [Amazon CloudWatch documentation](#)
- [Collect metrics and logs from Amazon EC2 instances and on-premises servers with the CloudWatch Agent](#)
- [Accessing Amazon CloudWatch Logs for AWS Lambda](#)
- [Using CloudWatch Logs with container instances](#)
- [Publish custom metrics](#)
- [AWS Answers: Centralized Logging](#)
- [AWS Services That Publish CloudWatch Metrics](#)
- [Monitoring Amazon EKS on AWS Fargate](#)

Related videos:

- [Application Performance Management on AWS](#)

Related examples:

- [Level 100: Monitoring with CloudWatch Dashboards](#)
- [Level 100: Monitoring Windows EC2 instance with CloudWatch Dashboards](#)
- [Level 100: Monitoring an Amazon Linux EC2 instance with CloudWatch Dashboards](#)

PERF02-BP04 Configure and right-size compute resources

Configure and right-size compute resources to match your workload's performance requirements and avoid under- or over-utilized resources.

Common anti-patterns:

- You ignore your workload performance requirements resulting in over-provisioned or under-provisioned compute resources.
- You only choose the largest or smallest instance available for all workloads.
- You only use one instance family for ease of management.
- You ignore recommendations from AWS Cost Explorer or Compute Optimizer for right-sizing.
- You do not re-evaluate the workload for suitability of new instance types.
- You certify only a small number of instance configurations for your organization.

Benefits of establishing this best practice: Right-sizing compute resources ensures optimal operation in the cloud by avoiding over-provisioning and under-provisioning resources. Properly sizing compute resources typically results in better performance and enhanced customer experience, while also lowering cost.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Right-sizing allows organizations to operate their cloud infrastructure in an efficient and cost-effective manner while addressing their business needs. Over-provisioning cloud resources can lead to extra costs, while under-provisioning can result in poor performance and a negative customer experience. AWS provides tools such as [AWS Compute Optimizer](#) and [AWS Trusted Advisor](#) that use historical data to provide recommendations to right-size your compute resources.

Implementation steps

- Choose an instance type to best fit your needs:
 - [How do I choose the appropriate Amazon EC2 instance type for my workload?](#)
 - [Attribute-based instance type selection for Amazon EC2 Fleet](#)
 - [Create an Auto Scaling group using attribute-based instance type selection](#)
 - [Optimizing your Kubernetes compute costs with Karpenter consolidation](#)

- Analyze the various performance characteristics of your workload and how these characteristics relate to memory, network, and CPU usage. Use this data to choose resources that best match your workload's profile and performance goals.
- Monitor your resource usage using AWS monitoring tools such as Amazon CloudWatch.
- Select the right configuration for compute resources.
 - For ephemeral workloads, evaluate [instance Amazon CloudWatch metrics](#) such as CPUUtilization to identify if the instance is under-utilized or over-utilized.
 - For stable workloads, check AWS rightsizing tools such as AWS Compute Optimizer and AWS Trusted Advisor at regular intervals to identify opportunities to optimize and right-size the compute resource.
 - [Well-Architected Lab - Rightsizing Recommendations](#)
 - [Well-Architected Lab - Rightsizing with Compute Optimizer](#)
- Test configuration changes in a non-production environment before implementing in a live environment.
- Continually re-evaluate new compute offerings and compare against your workload's needs.

Resources

Related documents:

- [Cloud Compute with AWS](#)
- [Amazon EC2 Instance Types](#)
- [Amazon ECS Containers: Amazon ECS Container Instances](#)
- [Amazon EKS Containers: Amazon EKS Worker Nodes](#)
- [Functions: Lambda Function Configuration](#)
- [Processor State Control for Your Amazon EC2 Instance](#)

Related videos:

- [Amazon EC2 foundations](#)
- [Better, faster, cheaper compute: Cost-optimizing Amazon EC2](#)
- [Deploy ML models for inference at high performance and low cost](#)
- [Optimize performance and cost for your AWS compute](#)
- [Powering next-gen Amazon EC2: Deep dive into the Nitro system](#)

- [Simplifying Data Processing to Enhance Innovation with Serverless Tools](#)

Related examples:

- [Rightsizing with Compute Optimizer and Memory utilization enabled](#)
- [AWS Compute Optimizer Demo code](#)

PERF02-BP05 Scale your compute resources dynamically

Use the elasticity of the cloud to scale your compute resources up or down dynamically to match your needs and avoid over- or under-provisioning capacity for your workload.

Common anti-patterns:

- You react to alarms by manually increasing capacity.
- You use the same sizing guidelines (generally static infrastructure) as in on-premises.
- You leave increased capacity after a scaling event instead of scaling back down.

Benefits of establishing this best practice: Configuring and testing the elasticity of compute resources can help you save money, maintain performance benchmarks, and improve reliability as traffic changes.

Level of risk exposed if this best practice is not established: High

Implementation guidance

AWS provides the flexibility to scale your resources up or down dynamically through a variety of scaling mechanisms in order to meet changes in demand. Combined with compute-related metrics, a dynamic scaling allows workloads to automatically respond to changes and use the optimal set of compute resources to achieve its goal.

You can use a number of different approaches to match supply of resources with demand.

- **Target-tracking approach:** Monitor your scaling metric and automatically increase or decrease capacity as you need it.
- **Predictive scaling:** Scale in anticipation of daily and weekly trends.
- **Schedule-based approach:** Set your own scaling schedule according to predictable load changes.
- **Service scaling:** Choose services (like serverless) that automatically scale by design.

You must ensure that workload deployments can handle both scale-up and scale-down events.

Implementation steps

- Compute instances, containers, and functions provide mechanisms for elasticity, either in combination with autoscaling or as a feature of the service. Here are some examples of automatic scaling mechanisms:

Autoscaling Mechanism	Where to use
Amazon EC2 Auto Scaling	To ensure you have the correct number of Amazon EC2 instances available to handle the user load for your application.
Application Auto Scaling	To automatically scale the resources for individual AWS services beyond Amazon EC2 such as AWS Lambda functions or Amazon Elastic Container Service (Amazon ECS) services.
Kubernetes Cluster Autoscaler/Karpenter	To automatically scale Kubernetes clusters.

- Scaling is often discussed related to compute services like Amazon EC2 Instances or AWS Lambda functions. Be sure to also consider the configuration of non-compute services like [AWS Glue](#) to match the demand.
- Verify that the metrics for scaling match the characteristics of the workload being deployed. If you are deploying a video transcoding application, 100% CPU utilization is expected and should not be your primary metric. Use the depth of the transcoding job queue instead. You can use a [customized metric](#) for your scaling policy if required. To choose the right metrics, consider the following guidance for Amazon EC2:
 - The metric should be a valid utilization metric and describe how busy an instance is.
 - The metric value must increase or decrease proportionally to the number of instances in the Auto Scaling group.
 - Make sure that you use [dynamic scaling](#) instead of [manual scaling](#) for your Auto Scaling group. We also recommend that you use [target tracking scaling policies](#) in your dynamic scaling.
 - Verify that workload deployments can handle both scaling events (up and down). As an example, you can use [Activity history](#) to verify a scaling activity for an Auto Scaling group.

- Evaluate your workload for predictable patterns and proactively scale as you anticipate predicted and planned changes in demand. With predictive scaling, you can eliminate the need to overprovision capacity. For more detail, see [Predictive Scaling with Amazon EC2 Auto Scaling](#).

Resources

Related documents:

- [Cloud Compute with AWS](#)
- [Amazon EC2 Instance Types](#)
- [Amazon ECS Containers: Amazon ECS Container Instances](#)
- [Amazon EKS Containers: Amazon EKS Worker Nodes](#)
- [Functions: Lambda Function Configuration](#)
- [Processor State Control for Your Amazon EC2 Instance](#)
- [Deep Dive on Amazon ECS Cluster Auto Scaling](#)
- [Introducing Karpenter – An Open-Source High-Performance Kubernetes Cluster Autoscaler](#)

Related videos:

- [Amazon EC2 foundations](#)
- [Better, faster, cheaper compute: Cost-optimizing Amazon EC2](#)
- [Optimize performance and cost for your AWS compute](#)
- [Powering next-gen Amazon EC2: Deep dive into the Nitro system](#)
- [Build a cost-, energy-, and resource-efficient compute environment](#)

Related examples:

- [Amazon EC2 Auto Scaling Group Examples](#)
- [Implement Autoscaling with Karpenter](#)

PERF02-BP06 Use optimized hardware-based compute accelerators

Use hardware accelerators to perform certain functions more efficiently than CPU-based alternatives.

Common anti-patterns:

- In your workload, you haven't benchmarked a general-purpose instance against a purpose-built instance that can deliver higher performance and lower cost.
- You are using hardware-based compute accelerators for tasks that can be more efficient using CPU-based alternatives.
- You are not monitoring GPU usage.

Benefits of establishing this best practice: By using hardware-based accelerators, such as graphics processing units (GPUs) and field programmable gate arrays (FPGAs), you can perform certain processing functions more efficiently.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Accelerated computing instances provide access to hardware-based compute accelerators such as GPUs and FPGAs. These hardware accelerators perform certain functions like graphic processing or data pattern matching more efficiently than CPU-based alternatives. Many accelerated workloads, such as rendering, transcoding, and machine learning, are highly variable in terms of resource usage. Only run this hardware for the time needed, and decommission them with automation when not required to improve overall performance efficiency.

Implementation steps

- Identify which [accelerated computing instances](#) can address your requirements.
- For machine learning workloads, take advantage of purpose-built hardware that is specific to your workload, such as [AWS Trainium](#), [AWS Inferentia](#), and [Amazon EC2 DL1](#). AWS Inferentia instances such as Inf2 instances [offer up to 50% better performance/watt over comparable Amazon EC2 instances](#).
- Collect usage metrics for your accelerated computing instances. For example, you can use CloudWatch agent to collect metrics such as utilization_gpu and utilization_memory for your GPUs as shown in [Collect NVIDIA GPU metrics with Amazon CloudWatch](#).
- Optimize the code, network operation, and settings of hardware accelerators to make sure that underlying hardware is fully utilized.
 - [Optimize GPU settings](#)
 - [GPU Monitoring and Optimization in the Deep Learning AMI](#)

- [Optimizing I/O for GPU performance tuning of deep learning training in Amazon SageMaker](#)
- Use the latest high performant libraries and GPU drivers.
- Use automation to release GPU instances when not in use.

Resources

Related documents:

- [GPU instances](#)
- [Instances with AWS Trainium](#)
- [Instances with AWS Inferentia](#)
- [Let's Architect! Architecting with custom chips and accelerators](#)
- [Accelerated Computing](#)
- [Amazon EC2 VT1 Instances](#)
- [How do I choose the appropriate Amazon EC2 instance type for my workload?](#)
- [Choose the best AI accelerator and model compilation for computer vision inference with Amazon SageMaker](#)

Related videos:

- [How to select Amazon EC2 GPU instances for deep learning](#)
- [Deploying Cost-Effective Deep Learning Inference](#)

Data management

Questions

- [PERF 3. How do you store, manage, and access data in your workload?](#)

PERF 3. How do you store, manage, and access data in your workload?

The optimal data management solution for a particular system varies based on the kind of data type (block, file, or object), access patterns (random or sequential), required throughput, frequency of access (online, offline, archival), frequency of update (WORM, dynamic), and availability and

durability constraints. Well-Architected workloads use purpose-built data stores which allow different features to improve performance.

Best practices

- [PERF03-BP01 Use a purpose-built data store that best supports your data access and storage requirements](#)
- [PERF03-BP02 Evaluate available configuration options for data store](#)
- [PERF03-BP03 Collect and record data store performance metrics](#)
- [PERF03-BP04 Implement strategies to improve query performance in data store](#)
- [PERF03-BP05 Implement data access patterns that utilize caching](#)

PERF03-BP01 Use a purpose-built data store that best supports your data access and storage requirements

Understand data characteristics (like shareable, size, cache size, access patterns, latency, throughput, and persistence of data) to select the right purpose-built data stores (storage or database) for your workload.

Common anti-patterns:

- You stick to one data store because there is internal experience and knowledge of one particular type of database solution.
- You assume that all workloads have similar data storage and access requirements.
- You have not implemented a data catalog to inventory your data assets.

Benefits of establishing this best practice: Understanding data characteristics and requirements allows you to determine the most efficient and performant storage technology appropriate for your workload needs.

Level of risk exposed if this best practice is not established: High

Implementation guidance

When selecting and implementing data storage, make sure that the querying, scaling, and storage characteristics support the workload data requirements. AWS provides numerous data storage and database technologies including block storage, object storage, streaming storage, file system, relational, key-value, document, in-memory, graph, time series, and ledger databases. Each data

management solution has options and configurations available to you to support your use-cases and data models. By understanding data characteristics and requirements, you can break away from monolithic storage technology and restrictive, one-size-fits-all approaches to focus on managing data appropriately.

Implementation steps

- Conduct an inventory of the various data types that exist in your workload.
- Understand and document data characteristics and requirements, including:
 - Data type (unstructured, semi-structured, relational)
 - Data volume and growth
 - Data durability: persistent, ephemeral, transient
 - ACID (atomicity, consistency, isolation, durability) requirements
 - Data access patterns (read-heavy or write-heavy)
 - Latency
 - Throughput
 - IOPS (input/output operations per second)
 - Data retention period
- Learn about different data stores available for your workload on AWS that can meet your data characteristics (as outlined in [PERF01-BP01 Learn about and understand available cloud services and features](#)). Some examples of AWS storage technologies and their key characteristics include:

Type	AWS Services	Key characteristics
Object storage	Amazon S3	Unlimited scalability, high availability, and multiple options for accessibility. Transferring and accessing objects in and out of Amazon S3 can use a service, such as Transfer Acceleration or Access Points , to support your location, security needs, and access patterns.

Type	AWS Services	Key characteristics
Archiving storage	Amazon S3 Glacier	Built for data archiving.
Streaming storage	Amazon Kinesis Amazon Managed Streaming for Apache Kafka (Amazon MSK)	Efficient ingestion and storage of streaming data.
Shared file system	Amazon Elastic File System (Amazon EFS)	Mountable file system that can be accessed by multiple types of compute solutions.
Shared file system	Amazon FSx	Built on the latest AWS compute solutions to support four commonly used file systems: NetApp ONTAP, OpenZFS, Windows File Server, and Lustre. Amazon FSx latency, throughput, and IOPS vary per file system and should be considered when selecting the right file system for your workload needs.
Block storage	Amazon Elastic Block Store (Amazon EBS)	Scalable, high-performance block-storage service designed for Amazon Elastic Compute Cloud (Amazon EC2). Amazon EBS includes SSD-backed storage for transactional, IOPS-intensive workloads and HDD-backed storage for throughput-intensive workloads.

Type	AWS Services	Key characteristics
Relational database	Amazon Aurora , Amazon RDS , Amazon Redshift .	Designed to support ACID (atomicity, consistency, isolation, durability) transactions, and maintain referential integrity and strong data consistency. Many traditional applications, enterprise resource planning (ERP), customer relationship management (CRM), and ecommerce use relational databases to store their data.
Key-value database	Amazon DynamoDB	Optimized for common access patterns, typically to store and retrieve large volumes of data. High-traffic web apps, ecommerce systems, and gaming applications are typical use-cases for key-value databases.
Document database	Amazon DocumentDB	Designed to store semi-structured data as JSON-like documents. These databases help developers build and update applications such as content management, catalogs, and user profiles quickly.

Type	AWS Services	Key characteristics
In-memory database	Amazon ElastiCache , Amazon MemoryDB for Redis	Used for applications that require real-time access to data, lowest latency and highest throughput. You may use in-memory databases for application caching, session management, gaming leaderboards, low latency ML feature store, microservices messaging system, and a high-throughput streaming mechanism
Graph database	Amazon Neptune	Used for applications that must navigate and query millions of relationships between highly connected graph datasets with millisecond latency at large scale. Many companies use graph databases for fraud detection, social networking, and recommendation engines.
Time Series database	Amazon Timestream	Used to efficiently collect, synthesize, and derive insights from data that changes over time. IoT applications, DevOps, and industrial telemetry can utilize time-series databases.

Type	AWS Services	Key characteristics
Wide column	Amazon Keyspaces (for Apache Cassandra)	Uses tables, rows, and columns, but unlike a relational database, the names and format of the columns can vary from row to row in the same table. You typically see a wide column store in high scale industrial apps for equipment maintenance, fleet management, and route optimization.
Ledger	Amazon Quantum Ledger Database (Amazon QLDB)	Provides a centralized and trusted authority to maintain a scalable, immutable, and cryptographically verifiable record of transactions for every application. We see ledger databases used for systems of record, supply chain, registrations, and even banking transactions.

- If you are building a data platform, leverage [modern data architecture](#) on AWS to integrate your data lake, data warehouse, and purpose-built data stores.
- The key questions that you need to consider when choosing a data store for your workload are as follows:

Question	Things to consider
How is the data structured?	<ul style="list-style-type: none"> • If the data is unstructured, consider an object-store such as Amazon S3 or a NoSQL database such as Amazon DocumentDB

Question	Things to consider
	<ul style="list-style-type: none">• For key-value data, consider DynamoDB, Amazon ElastiCache for Redis or Amazon MemoryDB for Redis
What level of referential integrity is required?	<ul style="list-style-type: none">• For foreign key constraints, relational databases such as Amazon RDS and Aurora can provide this level of integrity.• Typically, within a NoSQL data-model, you would de-normalize your data into a single document or collection of documents to be retrieved in a single request rather than joining across documents or tables.
Is ACID (atomicity, consistency, isolation, durability) compliance required?	<ul style="list-style-type: none">• If the ACID properties associated with relational databases are required, consider a relational database such as Amazon RDS and Aurora.• If strong consistency is required for NoSQL database, you can use strongly consistent reads with DynamoDB.
How will the storage requirements change over time? How does this impact scalability?	<ul style="list-style-type: none">• Serverless databases such as DynamoDB and Amazon Quantum Ledger Database (Amazon QLDB) will scale dynamically.• Relational databases have upper bounds on provisioned storage, and often must be horizontally partitioned using mechanisms such as sharding once they reach these limits.

Question	Things to consider
What is the proportion of read queries in relation to write queries? Would caching be likely to improve performance?	<ul style="list-style-type: none"> Read-heavy workloads can benefit from a caching layer, like ElastiCache or DAX if the database is DynamoDB. Reads can also be offloaded to read replicas with relational databases such as Amazon RDS.
Does storage and modification (OLTP - Online Transaction Processing) or retrieval and reporting (OLAP - Online Analytical Processing) have a higher priority?	<ul style="list-style-type: none"> For high-throughput read as-is transactional processing, consider a NoSQL database such as DynamoDB. For high-throughput and complex read patterns (like join) with consistency use Amazon RDS. For analytical queries, consider a columnar database such as Amazon Redshift or exporting the data to Amazon S3 and performing analytics using Athena or Amazon QuickSight.
What level of durability does the data require?	<ul style="list-style-type: none"> Aurora automatically replicates your data across three Availability Zones within a Region, meaning your data is highly durable with less chance of data loss. DynamoDB is automatically replicated across multiple Availability Zones, providing high availability and data durability. Amazon S3 provides 11 nines of durability. Many database services, such as Amazon RDS and DynamoDB, support exporting data to Amazon S3 for long-term retention and archival.

Question	Things to consider
Is there a desire to move away from commercial database engines or licensing costs?	<ul style="list-style-type: none"> Consider open-source engines such as PostgreSQL and MySQL on Amazon RDS or Aurora. Leverage AWS Database Migration Service and AWS Schema Conversion Tool to perform migrations from commercial database engines to open-source
What is the operational expectation for the database? Is moving to managed services a primary concern?	<ul style="list-style-type: none"> Leveraging Amazon RDS instead of Amazon EC2, and DynamoDB or Amazon DocumentDB instead of self-hosting a NoSQL database can reduce operational overhead.
How is the database currently accessed? Is it only application access, or are there business intelligence (BI) users and other connected off-the-shelf applications?	<ul style="list-style-type: none"> If you have dependencies on external tooling then you may have to maintain compatibility with the databases they support. Amazon RDS is fully compatible with the difference engine versions that it supports including Microsoft SQL Server, Oracle, MySQL, and PostgreSQL.

- Perform experiments and benchmarking in a non-production environment to identify which data store can address your workload requirements.

Resources

Related documents:

- [Amazon EBS Volume Types](#)
- [Amazon EC2 Storage](#)
- [Amazon EFS: Amazon EFS Performance](#)
- [Amazon FSx for Lustre Performance](#)
- [Amazon FSx for Windows File Server Performance](#)
- [Amazon S3 Glacier: S3 Glacier Documentation](#)

- [Amazon S3: Request Rate and Performance Considerations](#)
- [Cloud Storage with AWS](#)
- [Amazon EBS I/O Characteristics](#)
- [Cloud Databases with AWS](#)
- [AWS Database Caching](#)
- [DynamoDB Accelerator](#)
- [Amazon Aurora best practices](#)
- [Amazon Redshift performance](#)
- [Amazon Athena top 10 performance tips](#)
- [Amazon Redshift Spectrum best practices](#)
- [Amazon DynamoDB best practices](#)
- [Choose between Amazon EC2 and Amazon RDS](#)
- [Best Practices for Implementing Amazon ElastiCache](#)

Related videos:

- [Deep dive on Amazon EBS](#)
- [Optimize your storage performance with Amazon S3](#)
- [Modernize apps with purpose-built databases](#)
- [Amazon Aurora storage demystified: How it all works](#)
- [Amazon DynamoDB deep dive: Advanced design patterns](#)

Related examples:

- [Amazon EFS CSI Driver](#)
- [Amazon EBS CSI Driver](#)
- [Amazon EFS Utilities](#)
- [Amazon EBS Autoscale](#)
- [Amazon S3 Examples](#)
- [Optimize Data Pattern using Amazon Redshift Data Sharing](#)
- [Database Migrations](#)

- [MS SQL Server - AWS Database Migration Service \(AWS DMS\) Replication Demo](#)
- [Database Modernization Hands On Workshop](#)
- [Amazon Neptune Samples](#)

PERF03-BP02 Evaluate available configuration options for data store

Understand and evaluate the various features and configuration options available for your data stores to optimize storage space and performance for your workload.

Common anti-patterns:

- You only use one storage type, such as Amazon EBS, for all workloads.
- You use provisioned IOPS for all workloads without real-world testing against all storage tiers.
- You are not aware of the configuration options of your chosen data management solution.
- You rely solely on increasing instance size without looking at other available configuration options.
- You are not testing the scaling characteristics of your data store.

Benefits of establishing this best practice: By exploring and experimenting with the data store configurations, you may be able to reduce the cost of infrastructure, improve performance, and lower the effort required to maintain your workloads.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

A workload could have one or more data stores used based on data storage and access requirements. To optimize your performance efficiency and cost, you must evaluate data access patterns to determine the appropriate data store configurations. While you explore data store options, take into consideration various aspects such as the storage options, memory, compute, read replica, consistency requirements, connection pooling, and caching options. Experiment with these various configuration options to improve performance efficiency metrics.

Implementation steps

- Understand the current configurations (like instance type, storage size, or database engine version) of your data store.

- Review AWS documentation and best practices to learn about recommended configuration options that can help improve the performance of your data store. Key data store options to consider are the following:

Configuration option	Examples
Offloading reads (like read replicas and caching)	<ul style="list-style-type: none">• For DynamoDB tables, you can offload reads using DAX for caching.• You can create an Amazon ElastiCache for Redis cluster and configure your application to read from the cache first, falling back to the database if the requested item is not present.• Relational databases such as Amazon RDS and Aurora, and provisioned NoSQL databases such as Neptune and Amazon DocumentDB all support adding read replicas to offload the read portions of the workload.• Serverless databases such as DynamoDB will scale automatically. Ensure that you have enough read capacity units (RCU) provisioned to handle the workload.

Configuration option	Examples
Scaling writes (like partition key sharding or introducing a queue)	<ul style="list-style-type: none">For relational databases, you can increase the size of the instance to accommodate an increased workload or increase the provisioned IOPs to allow for an increased throughput to the underlying storage.You can also introduce a queue in front of your database rather than writing directly to the database. This pattern allows you to decouple the ingestion from the database and control the flow-rate so the database does not get overwhelmed.Batching your write requests rather than creating many short-lived transactions can help improve throughput in high-write volume relational databases.Serverless databases like DynamoDB can scale the write throughput automatically or by adjusting the provisioned write capacity units (WCUs) depending on the capacity mode.You can still run into issues with hot partitions when you reach the throughput limits for a given partition key. This can be mitigated by choosing a more evenly distributed partition key or by write-sharding the partition key.

Configuration option	Examples
Policies to manage the lifecycle of your datasets	<ul style="list-style-type: none"> You can use Amazon S3 Lifecycle to manage your objects throughout their lifecycle. If your access patterns are unknown, changing, or unpredictable, you can use Amazon S3 Intelligent-Tiering, which monitors access patterns and automatically moves objects that have not been accessed to lower-cost access tiers. You can leverage Amazon S3 Storage Lens metrics to identify optimization opportunities and gaps in lifecycle management. Amazon EFS lifecycle management automatically manages file storage for your file systems.
Connection management and pooling	<ul style="list-style-type: none"> Amazon RDS Proxy can be used with Amazon RDS and Aurora to manage connections to the database. Serverless databases such as DynamoDB do not have connections associated with them, but consider the provisioned capacity and automatic scaling policies to deal with spikes in load.

- Perform experiments and benchmarking in non-production environment to identify which configuration option can address your workload requirements.
- Once you have experimented, plan your migration and validate your performance metrics.
- Use AWS monitoring (like [Amazon CloudWatch](#)) and optimization (like [Amazon S3 Storage Lens](#)) tools to continuously optimize your data store using real-world usage pattern.

Resources

Related documents:

- [Cloud Storage with AWS](#)
- [Amazon EBS Volume Types](#)
- [Amazon EC2 Storage](#)
- [Amazon EFS: Amazon EFS Performance](#)
- [Amazon FSx for Lustre Performance](#)
- [Amazon FSx for Windows File Server Performance](#)
- [Amazon S3 Glacier: S3 Glacier Documentation](#)
- [Amazon S3: Request Rate and Performance Considerations](#)
- [Cloud Storage with AWS](#)
- [Cloud Storage with AWS](#)
- [Amazon EBS I/O Characteristics](#)
- [Cloud Databases with AWS](#)
- [AWS Database Caching](#)
- [DynamoDB Accelerator](#)
- [Amazon Aurora best practices](#)
- [Amazon Redshift performance](#)
- [Amazon Athena top 10 performance tips](#)
- [Amazon Redshift Spectrum best practices](#)
- [Amazon DynamoDB best practices](#)

Related videos:

- [Deep dive on Amazon EBS](#)
- [Optimize your storage performance with Amazon S3](#)
- [Modernize apps with purpose-built databases](#)
- [Amazon Aurora storage demystified: How it all works](#)
- [Amazon DynamoDB deep dive: Advanced design patterns](#)

Related examples:

- [Amazon EFS CSI Driver](#)

- [Amazon EBS CSI Driver](#)
- [Amazon EFS Utilities](#)
- [Amazon EBS Autoscale](#)
- [Amazon S3 Examples](#)
- [Amazon DynamoDB Examples](#)
- [AWS Database migration samples](#)
- [Database Modernization Workshop](#)
- [Working with parameters on your Amazon RDS for Postgress DB](#)

PERF03-BP03 Collect and record data store performance metrics

Track and record relevant performance metrics for your data store to understand how your data management solutions are performing. These metrics can help you optimize your data store, verify that your workload requirements are met, and provide a clear overview on how the workload performs.

Common anti-patterns:

- You only use manual log file searching for metrics.
- You only publish metrics to internal tools used by your team and don't have a comprehensive picture of your workload.
- You only use the default metrics recorded by your selected monitoring software.
- You only review metrics when there is an issue.
- You only monitor system-level metrics and do not capture data access or usage metrics.

Benefits of establishing this best practice: Establishing a performance baseline helps you understand the normal behavior and requirements of workloads. Abnormal patterns can be identified and debugged faster, improving the performance and reliability of the data store.

Level of risk exposed if this best practice is not established: High

Implementation guidance

To monitor the performance of your data stores, you must record multiple performance metrics over a period of time. This allows you to detect anomalies, as well as measure performance against business metrics to verify you are meeting your workload needs.

Metrics should include both the underlying system that is supporting the data store and the database metrics. The underlying system metrics might include CPU utilization, memory, available disk storage, disk I/O, cache hit ratio, and network inbound and outbound metrics, while the data store metrics might include transactions per second, top queries, average queries rates, response times, index usage, table locks, query timeouts, and number of connections open. This data is crucial to understand how the workload is performing and how the data management solution is used. Use these metrics as part of a data-driven approach to tune and optimize your workload's resources.

Use tools, libraries, and systems that record performance measurements related to database performance.

Implementation steps

1. Identify the key performance metrics for your data store to track.
 - a. [Amazon S3 Metrics and dimensions](#)
 - b. [Monitoring metrics for in an Amazon RDS instance](#)
 - c. [Monitoring DB load with Performance Insights on Amazon RDS](#)
 - d. [Overview of Enhanced Monitoring](#)
 - e. [DynamoDB Metrics and dimensions](#)
 - f. [Monitoring DynamoDB Accelerator](#)
 - g. [Monitoring Amazon MemoryDB for Redis with Amazon CloudWatch](#)
 - h. [Which Metrics Should I Monitor?](#)
 - i. [Monitoring Amazon Redshift cluster performance](#)
 - j. [Timestream metrics and dimensions](#)
 - k. [Amazon CloudWatch metrics for Amazon Aurora](#)
 - l. [Logging and monitoring in Amazon Keyspaces \(for Apache Cassandra\)](#)
 - m. [Monitoring Amazon Neptune Resources](#)
2. Use an approved logging and monitoring solution to collect these metrics. [Amazon CloudWatch](#) can collect metrics across the resources in your architecture. You can also collect and publish custom metrics to surface business or derived metrics. Use CloudWatch or third-party solutions to set alarms that indicate when thresholds are breached.
3. Check if data store monitoring can benefit from a machine learning solution that detects [performance anomalies](#).

- a. [Amazon DevOps Guru for Amazon RDS](#) provides visibility into performance issues and makes recommendations for corrective actions.
4. Configure data retention in your monitoring and logging solution to match your security and operational goals.
 - a. [Default data retention for CloudWatch metrics](#)
 - b. [Default data retention for CloudWatch Logs](#)

Resources

Related documents:

- [AWS Database Caching](#)
- [Amazon Athena top 10 performance tips](#)
- [Amazon Aurora best practices](#)
- [DynamoDB Accelerator](#)
- [Amazon DynamoDB best practices](#)
- [Amazon Redshift Spectrum best practices](#)
- [Amazon Redshift performance](#)
- [Cloud Databases with AWS](#)
- [Amazon RDS Performance Insights](#)

Related videos:

- [AWS purpose-built databases](#)
- [Amazon Aurora storage demystified: How it all works](#)
- [Amazon DynamoDB deep dive: Advanced design patterns](#)
- [Best Practices for Monitoring Redis Workloads on Amazon ElastiCache](#)

Related examples:

- [Level 100: Monitoring with CloudWatch Dashboards](#)
- [AWS Dataset Ingestion Metrics Collection Framework](#)
- [Amazon RDS Monitoring Workshop](#)

PERF03-BP04 Implement strategies to improve query performance in data store

Implement strategies to optimize data and improve data query to enable more scalability and efficient performance for your workload.

Common anti-patterns:

- You do not partition data in your data store.
- You store data in only one file format in your data store.
- You do not use indexes in your data store.

Benefits of establishing this best practice: Optimizing data and query performance results in more efficiency, lower cost, and improved user experience.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Data optimization and query tuning are critical aspects of performance efficiency in a data store, as they impact the performance and responsiveness of the entire cloud workload. Unoptimized queries can result in greater resource usage and bottlenecks, which reduce the overall efficiency of a data store.

Data optimization includes several techniques to ensure efficient data storage and access. This also helps to improve the query performance in a data store. Key strategies include data partitioning, data compression, and data denormalization, which help data to be optimized for both storage and access.

Implementation steps

- Understand and analyze the critical data queries which are performed in your data store.
- Identify the slow-running queries in your data store and use query plans to understand their current state.
 - [Analyzing the query plan in Amazon Redshift](#)
 - [Using EXPLAIN and EXPLAIN ANALYZE in Athena](#)
- Implement strategies to improve the query performance. Some of the key strategies include:
 - Using a [columnar file format](#) (like Parquet or ORC).
 - Compressing data in the data store to reduce storage space and I/O operation.

- Data partitioning to split data into smaller parts and reduce data scanning time.
 - [Partitioning data in Athena](#)
 - [Partitions and data distribution](#)
- Data indexing on the common columns in the query.
- Choose the right join operation for the query. When you join two tables, specify the larger table on the left side of join and the smaller table on the right side of the join.
- Distributed caching solution to improve latency and reduce the number of database I/O operation.
- Regular maintenance such as executing statistics.
- Experiment and test strategies in a non-production environment.

Resources

Related documents:

- [Amazon Aurora best practices](#)
- [Amazon Redshift performance](#)
- [Amazon Athena top 10 performance tips](#)
- [AWS Database Caching](#)
- [Best Practices for Implementing Amazon ElastiCache](#)
- [Partitioning data in Athena](#)

Related videos:

- [Optimize Data Pattern using Amazon Redshift Data Sharing](#)
- [Optimize Amazon Athena Queries with New Query Analysis Tools](#)

Related examples:

- [Amazon EFS CSI Driver](#)

PERF03-BP05 Implement data access patterns that utilize caching

Implement access patterns that can benefit from caching data for fast retrieval of frequently accessed data.

Common anti-patterns:

- You cache data that changes frequently.
- You rely on cached data as if it is durably stored and always available.
- You don't consider the consistency of your cached data.
- You don't monitor the efficiency of your caching implementation.

Benefits of establishing this best practice: Storing data in a cache can improve read latency, read throughput, user experience, and overall efficiency, as well as reduce costs.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

A cache is a software or hardware component aimed at storing data so that future requests for the same data can be served faster or more efficiently. The data stored in a cache can be reconstructed if lost by repeating an earlier calculation or fetching it from another data store.

Data caching can be one of the most effective strategies to improve your overall application performance and reduce burden on your underlying primary data sources. Data can be cached at multiple levels in the application, such as within the application making remote calls, known as *client-side caching*, or by using a fast secondary service for storing the data, known as *remote caching*.

Client-side caching

With client-side caching, each client (an application or service that queries the backend datastore) can store the results of their unique queries locally for a specified amount of time. This can reduce the number of requests across the network to a datastore by checking the local client cache first. If the results are not present, the application can then query the datastore and store those results locally. This pattern allows each client to store data in the closest location possible (the client itself), resulting in the lowest possible latency. Clients can also continue to serve some queries when the backend datastore is unavailable, increasing the availability of the overall system.

One disadvantage of this approach is that when multiple clients are involved, they may store the same cached data locally. This results in both duplicate storage usage and data inconsistency between those clients. One client might cache the results of a query, and one minute later another client can run the same query and get a different result.

Remote caching

To solve the issue of duplicate data between clients, a fast external service, or *remote cache*, can be used to store the queried data. Instead of checking a local data store, each client will check the remote cache before querying the backend datastore. This strategy allows for more consistent responses between clients, better efficiency in stored data, and a higher volume of cached data because the storage space scales independently of clients.

The disadvantage of a remote cache is that the overall system may see a higher latency, as an additional network hop is required to check the remote cache. Client-side caching can be used alongside remote caching for multi-level caching to improve the latency.

Implementation steps

1. Identify databases, APIs and network services that could benefit from caching. Services that have heavy read workloads, have a high read-to-write ratio, or are expensive to scale are candidates for caching.
 - [Database Caching](#)
 - [Enabling API caching to enhance responsiveness](#)
2. Identify the appropriate type of caching strategy that best fits your access pattern.
 - [Caching strategies](#)
 - [AWS Caching Solutions](#)
3. Follow [Caching Best Practices](#) for your data store.
4. Configure a cache invalidation strategy, such as a time-to-live (TTL), for all data that balances freshness of data and reducing pressure on backend datastore.
5. Enable features such as automatic connection retries, exponential backoff, client-side timeouts, and connection pooling in the client, if available, as they can improve performance and reliability.
 - [Best practices: Redis clients and Amazon ElastiCache for Redis](#)
6. Monitor cache hit rate with a goal of 80% or higher. Lower values may indicate insufficient cache size or an access pattern that does not benefit from caching.

- [Which metrics should I monitor?](#)
- [Best practices for monitoring Redis workloads on Amazon ElastiCache](#)
- [Monitoring best practices with Amazon ElastiCache for Redis using Amazon CloudWatch](#)

7. Implement [data replication](#) to offload reads to multiple instances and improve data read performance and availability.

Resources

Related documents:

- [Using the Amazon ElastiCache Well-Architected Lens](#)
- [Monitoring best practices with Amazon ElastiCache for Redis using Amazon CloudWatch](#)
- [Which Metrics Should I Monitor?](#)
- [Performance at Scale with Amazon ElastiCache whitepaper](#)
- [Caching challenges and strategies](#)

Related videos:

- [Amazon ElastiCache Learning Path](#)
- [Design for success with Amazon ElastiCache best practices](#)

Related examples:

- [Boosting MySQL database performance with Amazon ElastiCache for Redis](#)

Networking and content delivery

Questions

- [PERF 4. How do you select and configure networking resources in your workload?](#)

PERF 4. How do you select and configure networking resources in your workload?

The optimal networking solution for a workload varies based on latency, throughput requirements, jitter, and bandwidth. Physical constraints, such as user or on-premises resources, determine location options. These constraints can be offset with edge locations or resource placement.

Best practices

- [PERF04-BP01 Understand how networking impacts performance](#)
- [PERF04-BP02 Evaluate available networking features](#)
- [PERF04-BP03 Choose appropriate dedicated connectivity or VPN for your workload](#)
- [PERF04-BP04 Use load balancing to distribute traffic across multiple resources](#)
- [PERF04-BP05 Choose network protocols to improve performance](#)
- [PERF04-BP06 Choose your workload's location based on network requirements](#)
- [PERF04-BP07 Optimize network configuration based on metrics](#)

PERF04-BP01 Understand how networking impacts performance

Analyze and understand how network-related decisions impact your workload to provide efficient performance and improved user experience.

Common anti-patterns:

- All traffic flows through your existing data centers.
- You route all traffic through central firewalls instead of using cloud-native network security tools.
- You provision AWS Direct Connect connections without understanding actual usage requirements.
- You don't consider workload characteristics and encryption overhead when defining your networking solutions.
- You use on-premises concepts and strategies for networking solutions in the cloud.

Benefits of establishing this best practice: Understanding how networking impacts workload performance helps you identify potential bottlenecks, improve user experience, increase reliability, and lower operational maintenance as the workload changes.

Level of risk exposed if this best practice is not established: High

Implementation guidance

The network is responsible for the connectivity between application components, cloud services, edge networks, and on-premises data, and therefore it can heavily impact workload performance.

In addition to workload performance, user experience can be also impacted by network latency, bandwidth, protocols, location, network congestion, jitter, throughput, and routing rules.

Have a documented list of networking requirements from the workload including latency, packet size, routing rules, protocols, and supporting traffic patterns. Review the available networking solutions and identify which service meets your workload networking characteristics. Cloud-based networks can be quickly rebuilt, so evolving your network architecture over time is necessary to improve performance efficiency.

Implementation steps:

1. Define and document networking performance requirements, including metrics such as network latency, bandwidth, protocols, locations, traffic patterns (spikes and frequency), throughput, encryption, inspection, and routing rules.
2. Learn about key AWS networking services like [VPCs](#), [AWS Direct Connect](#), [Elastic Load Balancing \(ELB\)](#), and [Amazon Route 53](#).
3. Capture the following key networking characteristics:

Characteristics	Tools and metrics
Foundational networking characteristics	<ul style="list-style-type: none">• VPC Flow Logs• AWS Transit Gateway Flow Logs• AWS Transit Gateway metrics• AWS PrivateLink metrics
Application networking characteristics	<ul style="list-style-type: none">• Elastic Fabric Adapter• AWS App Mesh metrics• Amazon API Gateway metrics
Edge networking characteristics	<ul style="list-style-type: none">• Amazon CloudFront metrics• Amazon Route 53 metrics• AWS Global Accelerator metrics
Hybrid networking characteristics	<ul style="list-style-type: none">• AWS Direct Connect metrics• AWS Site-to-Site VPN metrics• AWS Client VPN metrics

Characteristics	Tools and metrics
Security networking characteristics	<ul style="list-style-type: none"> • AWS Cloud WAN metrics
Tracing characteristics	<ul style="list-style-type: none"> • AWS X-Ray • VPC Reachability Analyzer • Network Access Analyzer • Amazon Inspector • Amazon CloudWatch RUM

4. Benchmark and test network performance:

- a. [Benchmark](#) network throughput, as some factors can affect Amazon EC2 network performance when instances are in the same VPC. Measure the network bandwidth between Amazon EC2 Linux instances in the same VPC.
- b. Perform [load tests](#) to experiment with networking solutions and options.

Resources

Related documents:

- [Application Load Balancer](#)
- [EC2 Enhanced Networking on Linux](#)
- [EC2 Enhanced Networking on Windows](#)
- [EC2 Placement Groups](#)
- [Enabling Enhanced Networking with the Elastic Network Adapter \(ENA\) on Linux Instances](#)
- [Network Load Balancer](#)
- [Networking Products with AWS](#)
- [Transit Gateway](#)
- [Transitioning to latency-based routing in Amazon Route 53](#)
- [VPC Endpoints](#)
- [VPC Flow Logs](#)

Related videos:

- [Connectivity to AWS and hybrid AWS network architectures](#)
- [Optimizing Network Performance for Amazon EC2 Instances](#)
- [Improve Global Network Performance for Applications](#)
- [EC2 Instances and Performance Optimization Best Practices](#)
- [Optimizing Network Performance for Amazon EC2 Instances](#)
- [Networking best practices and tips with the Well-Architected Framework](#)
- [AWS networking best practices in large-scale migrations](#)

Related examples:

- [AWS Transit Gateway and Scalable Security Solutions](#)
- [AWS Networking Workshops](#)

PERF04-BP02 Evaluate available networking features

Evaluate networking features in the cloud that may increase performance. Measure the impact of these features through testing, metrics, and analysis. For example, take advantage of network-level features that are available to reduce latency, network distance, or jitter.

Common anti-patterns:

- You stay within one Region because that is where your headquarters is physically located.
- You use firewalls instead of security groups for filtering traffic.
- You break TLS for traffic inspection rather than relying on security groups, endpoint policies, and other cloud-native functionality.
- You only use subnet-based segmentation instead of security groups.

Benefits of establishing this best practice: Evaluating all service features and options can increase your workload performance, reduce the cost of infrastructure, decrease the effort required to maintain your workload, and increase your overall security posture. You can use the global AWS backbone to provide the optimal networking experience for your customers.

Level of risk exposed if this best practice is not established: High

Implementation guidance

AWS offers services like [AWS Global Accelerator](#) and [Amazon CloudFront](#) that can help improve network performance, while most AWS services have product features (such as the [Amazon S3 Transfer Acceleration](#) feature) to optimize network traffic.

Review which network-related configuration options are available to you and how they could impact your workload. Performance optimization depends on understanding how these options interact with your architecture and the impact that they will have on both measured performance and user experience.

Implementation steps

- Create a list of workload components.
 - Consider using [AWS Cloud WAN](#) to build, manage and monitor your organization's network when building a unified global network.
 - Monitor your global and core networks with [Amazon CloudWatch Logs metrics](#). Leverage [Amazon CloudWatch RUM](#), which provides insights to help to identify, understand, and enhance users' digital experience.
 - View aggregate network latency between AWS Regions and Availability Zones, as well as within each Availability Zone, using [AWS Network Manager](#) to gain insight into how your application performance relates to the performance of the underlying AWS network.
 - Use an existing configuration management database (CMDB) tool or a service such as [AWS Config](#) to create an inventory of your workload and how it's configured.
- If this is an existing workload, identify and document the benchmark for your performance metrics, focusing on the bottlenecks and areas to improve. Performance-related networking metrics will differ per workload based on business requirements and workload characteristics. As a start, these metrics might be important to review for your workload: bandwidth, latency, packet loss, jitter, and retransmits.
- If this is a new workload, perform [load tests](#) to identify performance bottlenecks.
- For the performance bottlenecks you identify, review the configuration options for your solutions to identify performance improvement opportunities. Check out the following key networking options and features:

Improvement opportunity	Solution
Network path or routes	Use Network Access Analyzer to identify paths or routes.
Network protocols	See PERF04-BP05 Choose network protocols to improve performance
Network topology	<p>Evaluate your operational and performance tradeoffs between VPC Peering and AWS Transit Gateway when connecting multiple accounts. AWS Transit Gateway simplifies how you interconnect all of your VPCs, which can span across thousands of AWS accounts and into on-premises networks. Share your AWS Transit Gateway between multiple accounts using AWS Resource Access Manager.</p> <p>See PERF04-BP03 Choose appropriate dedicated connectivity or VPN for your workload</p>

Improvement opportunity	Solution
Network services	<p>AWS Global Accelerator is a networking service that improves the performance of your users' traffic by up to 60% using the AWS global network infrastructure.</p> <p>Amazon CloudFront can improve the performance of your workload content delivery and latency globally.</p> <p>Use Lambda@edge to run functions that customize the content that CloudFront delivers closer to the users, reduce latency, and improve performance.</p> <p>Amazon Route 53 offers latency-based routing, geolocation routing, geoproximity routing, and IP-based routing options to help you improve your workload's performance for a global audience. Identify which routing option would optimize your workload performance by reviewing your workload traffic and user location when your workload is distributed globally.</p>

Improvement opportunity	Solution
Storage resource features	<p>Amazon S3 Transfer Acceleration is a feature that lets external users benefit from the networking optimizations of CloudFront to upload data to Amazon S3. This improves the ability to transfer large amounts of data from remote locations that don't have dedicated connectivity to the AWS Cloud.</p> <p>Amazon S3 Multi-Region Access Points replicates content to multiple Regions and simplifies the workload by providing one access point. When a Multi-Region Access Point is used, you can request or write data to Amazon S3 with the service identifying the lowest latency bucket.</p>

Improvement opportunity	Solution
Compute resource features	<p>Elastic Network Interfaces (ENA) used by Amazon EC2 instances, containers, and Lambda functions are limited on a per-flow basis. Review your placement groups to optimize your EC2 networking throughput. To avoid a bottleneck on a per flow-basis, design your application to use multiple flows. To monitor and get visibility into your compute related networking metrics, use CloudWatch Metrics and ethtool. The ethtool command is included in the ENA driver and exposes additional network-related metrics that can be published as a custom metric to CloudWatch.</p> <p>Amazon Elastic Network Adapters (ENA) provide further optimization by delivering better throughput for your instances within a cluster placement group.</p> <p>Elastic Fabric Adapter (EFA) is a network interface for Amazon EC2 instances that allows you to run workloads requiring high levels of internode communications at scale on AWS.</p> <p>Amazon EBS-optimized instances use an optimized configuration stack and provide additional, dedicated capacity to increase the Amazon EBS I/O.</p>

Resources

Related documents:

- [Amazon EBS - Optimized Instances](#)

- [Application Load Balancer](#)
- [EC2 Enhanced Networking on Linux](#)
- [EC2 Enhanced Networking on Windows](#)
- [EC2 Placement Groups](#)
- [Enabling Enhanced Networking with the Elastic Network Adapter \(ENA\) on Linux Instances](#)
- [Network Load Balancer](#)
- [Networking Products with AWS](#)
- [AWS Transit Gateway](#)
- [Transitioning to Latency-Based Routing in Amazon Route 53](#)
- [VPC Endpoints](#)
- [VPC Flow Logs](#)

Related videos:

- [Connectivity to AWS and hybrid AWS network architectures](#)
- [Optimizing Network Performance for Amazon EC2 Instances](#)
- [AWS Global Accelerator](#)

Related examples:

- [AWS Transit Gateway and Scalable Security Solutions](#)
- [AWS Networking Workshops](#)

PERF04-BP03 Choose appropriate dedicated connectivity or VPN for your workload

When hybrid connectivity is required to connect on-premises and cloud resources, provision adequate bandwidth to meet your performance requirements. Estimate the bandwidth and latency requirements for your hybrid workload. These numbers will drive your sizing requirements.

Common anti-patterns:

- You only evaluate VPN solutions for your network encryption requirements.
- You do not evaluate backup or redundant connectivity options.

- You do not identify all workload requirements (encryption, protocol, bandwidth, and traffic needs).

Benefits of establishing this best practice: Selecting and configuring appropriate connectivity solutions will increase the reliability of your workload and maximize performance. By identifying workload requirements, planning ahead, and evaluating hybrid solutions, you can minimize expensive physical network changes and operational overhead while increasing your time-to-value.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Develop a hybrid networking architecture based on your bandwidth requirements. [AWS Direct Connect](#) allows you to connect your on-premises network privately with AWS. It is suitable when you need high-bandwidth and low-latency while achieving consistent performance. A VPN connection establishes secure connection over the internet. It is used when only a temporary connection is required, when cost is a factor, or as a contingency while waiting for resilient physical network connectivity to be established when using AWS Direct Connect.

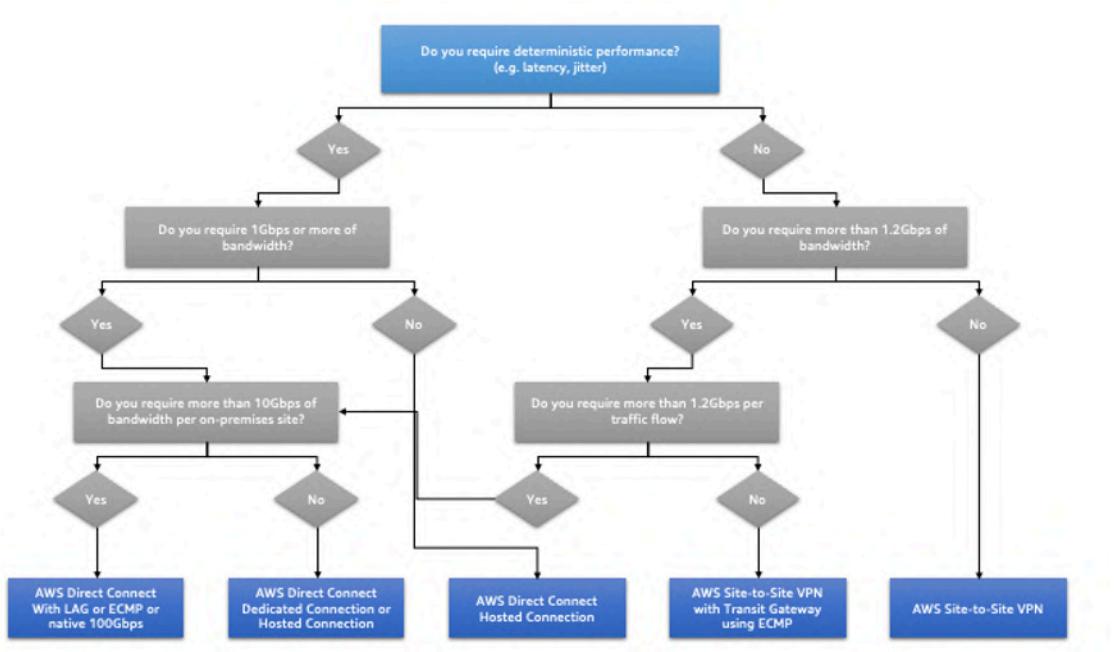
If your bandwidth requirements are high, you might consider multiple AWS Direct Connect or VPN services. Traffic can be load balanced across services, although we don't recommend load balancing between AWS Direct Connect and VPN because of the latency and bandwidth differences.

Implementation steps

1. Estimate the bandwidth and latency requirements of your existing applications.
 - a. For existing workloads that are moving to AWS, leverage the data from your internal network monitoring systems.
 - b. For new or existing workloads for which you don't have monitoring data, consult with the product owners to determine adequate performance metrics and provide a good user experience.
2. Select dedicated connection or VPN as your connectivity option. Based on all workload requirements (encryption, bandwidth, and traffic needs), you can either choose AWS Direct Connect or [AWS VPN](#) (or both). The following diagram can help you choose the appropriate connection type.
 - a. [AWS Direct Connect](#) provides dedicated connectivity to the AWS environment, from 50 Mbps up to 100 Gbps, using either dedicated connections or hosted connections. This gives you managed and controlled latency and provisioned bandwidth so your workload can connect

efficiently to other environments. Using AWS Direct Connect partners, you can have end-to-end connectivity from multiple environments, providing an extended network with consistent performance. AWS offers scaling direct connect connection bandwidth using either native 100 Gbps, link aggregation group (LAG), or BGP equal-cost multipath (ECMP).

- b. The AWS [Site-to-Site VPN](#) provides a managed VPN service supporting internet protocol security (IPsec). When a VPN connection is created, each VPN connection includes two tunnels for high availability.
3. Follow AWS documentation to choose an appropriate connectivity option:
 - a. If you decide to use AWS Direct Connect, select the appropriate bandwidth for your connectivity.
 - b. If you are using an AWS Site-to-Site VPN across multiple locations to connect to an AWS Region, use an [accelerated Site-to-Site VPN connection](#) for the opportunity to improve network performance.
 - c. If your network design consists of IPSec VPN connection over [AWS Direct Connect](#), consider using Private IP VPN to improve security and achieve segmentation. [AWS Site-to-Site Private IP VPN](#) is deployed on top of transit virtual interface (VIF).
 - d. [AWS Direct Connect SiteLink](#) allows creating low-latency and redundant connections between your data centers worldwide by sending data over the fastest path between [AWS Direct Connect locations](#), bypassing AWS Regions.
4. Validate your connectivity setup before deploying to production. Perform security and performance testing to assure it meets your bandwidth, reliability, latency, and compliance requirements.
5. Regularly monitor your connectivity performance and usage and optimize if required.



Deterministic performance flowchart

Resources

Related documents:

- [Network Load Balancer](#)
- [Networking Products with AWS](#)
- [AWS Transit Gateway](#)
- [Transitioning to latency-based Routing in Amazon Route 53](#)
- [VPC Endpoints](#)
- [Site-to-Site VPN](#)
- [Building a Scalable and Secure Multi-VPC AWS Network Infrastructure](#)
- [AWS Direct Connect](#)
- [Client VPN](#)

Related videos:

- [Connectivity to AWS and hybrid AWS network architectures](#)
- [Optimizing Network Performance for Amazon EC2 Instances](#)

- [AWS Global Accelerator](#)
- [AWS Direct Connect](#)
- [AWS Transit Gateway Connect](#)
- [VPN Solutions](#)
- [Security with VPN Solutions](#)

Related examples:

- [AWS Transit Gateway and Scalable Security Solutions](#)
- [AWS Networking Workshops](#)

PERF04-BP04 Use load balancing to distribute traffic across multiple resources

Distribute traffic across multiple resources or services to allow your workload to take advantage of the elasticity that the cloud provides. You can also use load balancing for offloading encryption termination to improve performance, reliability and manage and route traffic effectively.

Common anti-patterns:

- You don't consider your workload requirements when choosing the load balancer type.
- You don't leverage the load balancer features for performance optimization.
- The workload is exposed directly to the internet without a load balancer.
- You route all internet traffic through existing load balancers.
- You use generic TCP load balancing and making each compute node handle SSL encryption.

Benefits of establishing this best practice: A load balancer handles the varying load of your application traffic in a single Availability Zone or across multiple Availability Zones and enables high availability, automatic scaling, and better utilization for your workload.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Load balancers act as the entry point for your workload, from which point they distribute the traffic to your backend targets, such as compute instances or containers, to improve utilization.

Choosing the right load balancer type is the first step to optimize your architecture. Start by listing your workload characteristics, such as protocol (like TCP, HTTP, TLS, or WebSockets), target type (like instances, containers, or serverless), application requirements (like long running connections, user authentication, or stickiness), and placement (like Region, Local Zone, Outpost, or zonal isolation).

AWS provides multiple models for your applications to use load balancing. [Application Load Balancer](#) is best suited for load balancing of HTTP and HTTPS traffic and provides advanced request routing targeted at the delivery of modern application architectures, including microservices and containers.

[Network Load Balancer](#) is best suited for load balancing of TCP traffic where extreme performance is required. It is capable of handling millions of requests per second while maintaining ultra-low latencies, and it is optimized to handle sudden and volatile traffic patterns.

[Elastic Load Balancing](#) provides integrated certificate management and SSL/TLS decryption, allowing you the flexibility to centrally manage the SSL settings of the load balancer and offload CPU intensive work from your workload.

After choosing the right load balancer, you can start leveraging its features to reduce the amount of effort your backend has to do to serve the traffic.

For example, using both Application Load Balancer (ALB) and Network Load Balancer (NLB), you can perform SSL/TLS encryption offloading, which is an opportunity to avoid the CPU-intensive TLS handshake from being completed by your targets and also to improve certificate management.

When you configure SSL/TLS offloading in your load balancer, it becomes responsible for the encryption of the traffic from and to clients while delivering the traffic unencrypted to your backends, freeing up your backend resources and improving the response time for the clients.

Application Load Balancer can also serve HTTP/2 traffic without needing to support it on your targets. This simple decision can improve your application response time, as HTTP/2 uses TCP connections more efficiently.

Your workload latency requirements should be considered when defining the architecture. As an example, if you have a latency-sensitive application, you may decide to use Network Load Balancer, which offers extremely low latencies. Alternatively, you may decide to bring your workload closer to your customers by leveraging Application Load Balancer in [AWS Local Zones](#) or even [AWS Outposts](#).

Another consideration for latency-sensitive workloads is cross-zone load balancing. With cross-zone load balancing, each load balancer node distributes traffic across the registered targets in all allowed Availability Zones.

Use Auto Scaling integrated with your load balancer. One of the key aspects of a performance efficient system has to do with right-sizing your backend resources. To do this, you can leverage load balancer integrations for backend target resources. Using the load balancer integration with Auto Scaling groups, targets will be added or removed from the load balancer as required in response to incoming traffic. Load balancers can also integrate with [Amazon ECS](#) and [Amazon EKS](#) for containerized workloads.

- [Amazon ECS - Service load balancing](#)
- [Application load balancing on Amazon EKS](#)
- [Network load balancing on Amazon EKS](#)

Implementation steps

- Define your load balancing requirements including traffic volume, availability and application scalability.
- Choose the right load balancer type for your application.
 - Use Application Load Balancer for HTTP/HTTPS workloads.
 - Use Network Load Balancer for non-HTTP workloads that run on TCP or UDP.
 - Use a combination of both ([ALB as a target of NLB](#)) if you want to leverage features of both products. For example, you can do this if you want to use the static IPs of NLB together with HTTP header based routing from ALB, or if you want to expose your HTTP workload to an [AWS PrivateLink](#).
 - For a full comparison of load balancers, see [ELB product comparison](#).
- Use SSL/TLS offloading if possible.
 - Configure HTTPS/TLS listeners with both [Application Load Balancer](#) and [Network Load Balancer](#) integrated with [AWS Certificate Manager](#).
 - Note that some workloads may require end-to-end encryption for compliance reasons. In this case, it is a requirement to allow encryption at the targets.
 - For security best practices, see [SEC09-BP02 Enforce encryption in transit](#).
- Select the right routing algorithm (only ALB).

- The routing algorithm can make a difference in how well-used your backend targets are and therefore how they impact performance. For example, ALB provides [two options for routing algorithms](#):
- **Least outstanding requests:** Use to achieve a better load distribution to your backend targets for cases when the requests for your application vary in complexity or your targets vary in processing capability.
- **Round robin:** Use when the requests and targets are similar, or if you need to distribute requests equally among targets.
- Consider cross-zone or zonal isolation.
 - Use cross-zone turned off (zonal isolation) for latency improvements and zonal failure domains. It is turned off by default in NLB and in [ALB you can turn it off per target group](#).
 - Use cross-zone turned on for increased availability and flexibility. By default, cross-zone is turned on for ALB and in [NLB you can turn it on per target group](#).
- Turn on HTTP keep-alives for your HTTP workloads (only ALB). With this feature, the load balancer can reuse backend connections until the keep-alive timeout expires, improving your HTTP request and response time and also reducing resource utilization on your backend targets. For detail on how to do this for Apache and Nginx, see [What are the optimal settings for using Apache or NGINX as a backend server for ELB?](#)
- Turn on monitoring for your load balancer.
 - Turn on access logs for your [Application Load Balancer](#) and [Network Load Balancer](#).
 - The main fields to consider for ALB are `request_processing_time`, `request_processing_time`, and `response_processing_time`.
 - The main fields to consider for NLB are `connection_time` and `tls_handshake_time`.
 - Be ready to query the logs when you need them. You can use Amazon Athena to query both [ALB logs](#) and [NLB logs](#).
 - Create alarms for performance related metrics such as [TargetResponseTime for ALB](#).

Resources

Related documents:

- [ELB product comparison](#)
- [AWS Global Infrastructure](#)

- [Improving Performance and Reducing Cost Using Availability Zone Affinity](#)
- [Step by step for Log Analysis with Amazon Athena](#)
- [Querying Application Load Balancer logs](#)
- [Monitor your Application Load Balancers](#)
- [Monitor your Network Load Balancer](#)
- [Use Elastic Load Balancing to distribute traffic across the instances in your Auto Scaling group](#)

Related videos:

- [AWS re:Invent 2018: Elastic Load Balancing: Deep Dive and Best Practices](#)
- [AWS re:Invent 2021 - How to choose the right load balancer for your AWS workloads](#)
- [AWS re:Inforce 2022 - How to use Elastic Load Balancing to enhance your security posture at scale](#)
- [AWS re:Invent 2019: Get the most from Elastic Load Balancing for different workloads](#)

Related examples:

- [CDK and AWS CloudFormation samples for Log Analysis with Amazon Athena](#)

PERF04-BP05 Choose network protocols to improve performance

Make decisions about protocols for communication between systems and networks based on the impact to the workload's performance.

There is a relationship between latency and bandwidth to achieve throughput. If your file transfer is using Transmission Control Protocol (TCP), higher latencies will most likely reduce overall throughput. There are approaches to fix this with TCP tuning and optimized transfer protocols, but one solution is to use User Datagram Protocol (UDP).

Common anti-patterns:

- You use TCP for all workloads regardless of performance requirements.

Benefits of establishing this best practice: Verifying that an appropriate protocol is used for communication between users and workload components helps improve overall user experience for your applications. For instance, connection-less UDP allows for high speed, but it doesn't offer

retransmission or high reliability. TCP is a full featured protocol, but it requires greater overhead for processing the packets.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

If you have the ability to choose different protocols for your application and you have expertise in this area, optimize your application and end-user experience by using a different protocol. Note that this approach comes with significant difficulty and should only be attempted if you have optimized your application in other ways first.

A primary consideration for improving your workload's performance is to understand the latency and throughput requirements, and then choose network protocols that optimize performance.

When to consider using TCP

TCP provides reliable data delivery, and can be used for communication between workload components where reliability and guaranteed delivery of data is important. Many web-based applications rely on TCP-based protocols, such as HTTP and HTTPS, to open TCP sockets for communication between application components. Email and file data transfer are common applications that also make use of TCP, as it is a simple and reliable transfer mechanism between application components. Using TLS with TCP can add some overhead to the communication, which can result in increased latency and reduced throughput, but it comes with the advantage of security. The overhead comes mainly from the added overhead of the handshake process, which can take several round-trips to complete. Once the handshake is complete, the overhead of encrypting and decrypting data is relatively small.

When to consider using UDP

UDP is a connection-less-oriented protocol and is therefore suitable for applications that need fast, efficient transmission, such as log, monitoring, and VoIP data. Also, consider using UDP if you have workload components that respond to small queries from large numbers of clients to ensure optimal performance of the workload. Datagram Transport Layer Security (DTLS) is the UDP equivalent of Transport Layer Security (TLS). When using DTLS with UDP, the overhead comes from encrypting and decrypting the data, as the handshake process is simplified. DTLS also adds a small amount of overhead to the UDP packets, as it includes additional fields to indicate the security parameters and to detect tampering.

When to consider using SRD

Scalable reliable datagram (SRD) is a network transport protocol optimized for high-throughput workloads due to its ability to load-balancer traffic across multiple paths and quickly recover from packet drops or link failures. SRD is therefore best used for high performance computing (HPC) workloads that require high throughput and low latency communication between compute nodes. This might include parallel processing tasks such as simulation, modelling, and data analysis that involve a large amount of data transfer between nodes.

Implementation steps

1. Use the [AWS Global Accelerator](#) and [AWS Transfer Family](#) services to improve the throughput of your online file transfer applications. The AWS Global Accelerator service helps you achieve lower latency between your client devices and your workload on AWS. With AWS Transfer Family, you can use TCP-based protocols such as Secure Shell File Transfer Protocol (SFTP) and File Transfer Protocol over SSL (FTPS) to securely scale and manage your file transfers to AWS storage services.
2. Use network latency to determine if TCP is appropriate for communication between workload components. If the network latency between your client application and server is high, then the TCP three-way handshake can take some time, thereby impacting on the responsiveness of your application. Metrics such as time to first byte (TTFB) and round-trip time (RTT) can be used to measure network latency. If your workload serves dynamic content to users, consider using [Amazon CloudFront](#), which establishes a persistent connection to each origin for dynamic content to remove the connection setup time that would otherwise slow down each client request.
3. Using TLS with TCP or UDP can result in increased latency and reduced throughput for your workload due to the impact of encryption and decryption. For such workloads, consider SSL/TLS offloading on [Elastic Load Balancing](#) to improve workload performance by allowing the load balancer to handle SSL/TLS encryption and decryption process instead of having backend instances do it. This can help reduce the CPU utilization on the backend instances, which can improve performance and increase capacity.
4. Use the [Network Load Balancer \(NLB\)](#) to deploy services that rely on the UDP protocol, such as authentication and authorization, logging, DNS, IoT, and streaming media, to improve the performance and reliability of your workload. The NLB distributes incoming UDP traffic across multiple targets, allowing you to scale your workload horizontally, increase capacity, and reduce the overhead of a single target.
5. For your High Performance Computing (HPC) workloads, consider using the [Elastic Network Adapter \(ENA\) Express](#) functionality that uses the SRD protocol to improve network performance

by providing a higher single flow bandwidth (25 Gbps) and lower tail latency (99.9 percentile) for network traffic between EC2 instances.

6. Use the [Application Load Balancer \(ALB\)](#) to route and load balance your gRPC (Remote Procedure Calls) traffic between workload components or between gRPC clients and services. gRPC uses the TCP-based HTTP/2 protocol for transport and it provides performance benefits such as lighter network footprint, compression, efficient binary serialization, support for numerous languages, and bi-directional streaming.

Resources

Related documents:

- [Amazon EBS - Optimized Instances](#)
- [Application Load Balancer](#)
- [EC2 Enhanced Networking on Linux](#)
- [EC2 Enhanced Networking on Windows](#)
- [EC2 Placement Groups](#)
- [Enabling Enhanced Networking with the Elastic Network Adapter \(ENA\) on Linux Instances](#)
- [Network Load Balancer](#)
- [Networking Products with AWS](#)
- [AWS Transit Gateway](#)
- [Transitioning to Latency-Based Routing in Amazon Route 53](#)
- [VPC Endpoints](#)
- [VPC Flow Logs](#)

Related videos:

- [Connectivity to AWS and hybrid AWS network architectures](#)
- [Optimizing Network Performance for Amazon EC2 Instances](#)

Related examples:

- [AWS Transit Gateway and Scalable Security Solutions](#)

- [AWS Networking Workshops](#)

PERF04-BP06 Choose your workload's location based on network requirements

Evaluate options for resource placement to reduce network latency and improve throughput, providing an optimal user experience by reducing page load and data transfer times.

Common anti-patterns:

- You consolidate all workload resources into one geographic location.
- You chose the closest Region to your location but not to the workload end user.

Benefits of establishing this best practice: User experience is greatly affected by the latency between the user and your application. By using appropriate AWS Regions and the AWS private global network, you can reduce latency and deliver a better experience to remote users.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Resources, such as Amazon EC2 instances, are placed into Availability Zones within [AWS Regions](#), [AWS Local Zones](#), [AWS Outposts](#), or [AWS Wavelength](#) zones. Selection of this location influences network latency and throughput from a given user location. Edge services like [Amazon CloudFront](#) and [AWS Global Accelerator](#) can also be used to improve network performance by either caching content at edge locations or providing users with an optimal path to the workload through the AWS global network.

Amazon EC2 provides placement groups for networking. A placement group is a logical grouping of instances to decrease latency. Using placement groups with supported instance types and an Elastic Network Adapter (ENA) enables workloads to participate in a low-latency, reduced jitter 25 Gbps network. Placement groups are recommended for workloads that benefit from low network latency, high network throughput, or both.

Latency-sensitive services are delivered at edge locations using AWS global network, such as [Amazon CloudFront](#). These edge locations commonly provide services like content delivery network (CDN) and domain name system (DNS). By having these services at the edge, workloads can respond with low latency to requests for content or DNS resolution. These services also provide geographic services, such as geotargeting of content (providing different content based on the

end users' location) or latency-based routing to direct end users to the nearest Region (minimum latency).

Use edge services to reduce latency and to enable content caching. Configure cache control correctly for both DNS and HTTP/HTTPS to gain the most benefit from these approaches.

Implementation steps

- Capture information about the IP traffic going to and from network interfaces.
 - [Logging IP traffic using VPC Flow Logs](#)
 - [How the client IP address is preserved in AWS Global Accelerator](#)
- Analyze network access patterns in your workload to identify how users use your application.
 - Use monitoring tools, such as [Amazon CloudWatch](#) and [AWS CloudTrail](#), to gather data on network activities.
 - Analyze the data to identify the network access pattern.
- Select Regions for your workload deployment based on the following key elements:
 - **Where your data is located:** For data-heavy applications (such as big data and machine learning), application code should run as close to the data as possible.
 - **Where your users are located:** For user-facing applications, choose a Region (or Regions) close to your workload's users.
 - **Other constraints:** Consider constraints such as cost and compliance as explained in [What to Consider when Selecting a Region for your Workloads](#).
- Use [AWS Local Zones](#) to run workloads like video rendering. Local Zones allow you to benefit from having compute and storage resources closer to end users.
- Use [AWS Outposts](#) for workloads that need to remain on-premises and where you want that workload to run seamlessly with the rest of your other workloads in AWS.
- Applications like high-resolution live video streaming, high-fidelity audio, and augmented reality or virtual reality (AR/VR) require ultra-low-latency for 5G devices. For such applications, consider [AWS Wavelength](#). AWS Wavelength embeds AWS compute and storage services within 5G networks, providing mobile edge computing infrastructure for developing, deploying, and scaling ultra-low-latency applications.
- Use local caching or [AWS Caching Solutions](#) for frequently used assets to improve performance, reduce data movement, and lower environmental impact.

Service	When to use
Amazon CloudFront	Use to cache static content such as images, scripts, and videos, as well as dynamic content such as API responses or web applications.
Amazon ElastiCache	Use to cache content for web applications.
DynamoDB Accelerator	Use to add in-memory acceleration to your DynamoDB tables.

- Use services that can help you run code closer to users of your workload like the following:

Service	When to use
Lambda@edge	Use for compute-heavy operations that are initiated when objects are not in the cache.
Amazon CloudFront Functions	Use for simple use cases like HTTP(s) requests or response manipulations that can be initiated by short-lived functions.
AWS IoT Greengrass	Use to run local compute, messaging, and data caching for connected devices.

- Some applications require fixed entry points or higher performance by reducing first byte latency and jitter, and increasing throughput. These applications can benefit from networking services that provide static anycast IP addresses and TCP termination at edge locations. [AWS Global Accelerator](#) can improve performance for your applications by up to 60% and provide quick failover for multi-region architectures. AWS Global Accelerator provides you with static anycast IP addresses that serve as a fixed entry point for your applications hosted in one or more AWS Regions. These IP addresses permit traffic to ingress onto the AWS global network as close to your users as possible. AWS Global Accelerator reduces the initial connection setup time by establishing a TCP connection between the client and the AWS edge location closest to the client. Review the use of AWS Global Accelerator to improve the performance of your TCP/UDP workloads and provide quick failover for multi-Region architectures.

Resources

Related best practices:

- [COST07-BP02 Implement Regions based on cost](#)
- [COST08-BP03 Implement services to reduce data transfer costs](#)
- [REL10-BP01 Deploy the workload to multiple locations](#)
- [REL10-BP02 Select the appropriate locations for your multi-location deployment](#)
- [SUS01-BP01 Choose Region based on both business requirements and sustainability goals](#)
- [SUS02-BP04 Optimize geographic placement of workloads based on their networking requirements](#)
- [SUS04-BP07 Minimize data movement across networks](#)

Related documents:

- [AWS Global Infrastructure](#)
- [AWS Local Zones and AWS Outposts, choosing the right technology for your edge workload](#)
- [Placement groups](#)
- [AWS Local Zones](#)
- [AWS Outposts](#)
- [AWS Wavelength](#)
- [Amazon CloudFront](#)
- [AWS Global Accelerator](#)
- [AWS Direct Connect](#)
- [AWS Site-to-Site VPN](#)
- [Amazon Route 53](#)

Related videos:

- [AWS Local Zones Explainer Video](#)
- [AWS Outposts: Overview and How it Works](#)
- [AWS re:Invent 2021 - AWS Outposts: Bringing the AWS experience on premises](#)
- [AWS re:Invent 2020: AWS Wavelength: Run apps with ultra-low latency at 5G edge](#)

- [AWS re:Invent 2022 - AWS Local Zones: Building applications for a distributed edge](#)
- [AWS re:Invent 2021 - Building low-latency websites with Amazon CloudFront](#)
- [AWS re:Invent 2022 - Improve performance and availability with AWS Global Accelerator](#)
- [AWS re:Invent 2022 - Build your global wide area network using AWS](#)
- [AWS re:Invent 2020: Global traffic management with Amazon Route 53](#)

Related examples:

- [AWS Global Accelerator Workshop](#)
- [Handling Rewrites and Redirects using Edge Functions](#)

PERF04-BP07 Optimize network configuration based on metrics

Use collected and analyzed data to make informed decisions about optimizing your network configuration.

Common anti-patterns:

- You assume that all performance-related issues are application-related.
- You only test your network performance from a location close to where you have deployed the workload.
- You use default configurations for all network services.
- You overprovision the network resource to provide sufficient capacity.

Benefits of establishing this best practice: Collecting necessary metrics of your AWS network and implementing network monitoring tools allows you to understand network performance and optimize network configurations.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Monitoring traffic to and from VPCs, subnets, or network interfaces is crucial to understand how to utilize AWS network resources and optimize network configurations. By using the following AWS networking tools, you can further inspect information about the traffic usage, network access and logs.

Implementation steps

- Identify the key performance metrics such as latency or packet loss to collect. AWS provides several tools that can help you to collect these metrics. By using the following tools, you can further inspect information about the traffic usage, network access, and logs:

AWS tool	Where to use
Amazon VPC IP Address Manager.	Use IPAM to plan, track, and monitor IP addresses for your AWS and on-premises workloads. This is a best practice to optimize IP address usage and allocation.
VPC Flow logs	Use VPC Flow Logs to capture detailed information about traffic to and from network interfaces in your VPCs. With VPC Flow Logs, you can diagnose overly restrictive or permissive security group rules and determine the direction of the traffic to and from the network interfaces.
AWS Transit Gateway Flow Logs	Use AWS Transit Gateway Flow Logs to capture information about the IP traffic going to and from your transit gateways.
DNS query logging	Log information about public or private DNS queries Route 53 receives. With DNS logs, you can optimize DNS configurations by understanding the domain or subdomain that was requested or Route 53 EDGE locations that responded to DNS queries.

AWS tool	Where to use
<u>Reachability Analyzer</u>	<p>Reachability Analyzer helps you analyze and debug network reachability. Reachability Analyzer is a configuration analysis tool that allows you to perform connectivity testing between a source resource and a destination resource in your VPCs. This tool helps you verify that your network configuration matches your intended connectivity.</p>
<u>Network Access Analyzer</u>	<p>Network Access Analyzer helps you understand network access to your resources. You can use Network Access Analyzer to specify your network access requirements and identify potential network paths that do not meet your specified requirements. By optimizing your corresponding network configuration, you can understand and verify the state of your network and demonstrate if your network on AWS meets your compliance requirements.</p>
<u>Amazon CloudWatch</u>	<p>Use <u>Amazon CloudWatch</u> and turn on the appropriate metrics for network options. Make sure to choose the right network metric for your workload. For example, you can turn on metrics for VPC Network Address Usage, VPC NAT Gateway, AWS Transit Gateway, VPN tunnel, AWS Network Firewall, Elastic Load Balancing, and AWS Direct Connect. Continually monitoring metrics is a good practice to observe and understand your network status and usage, which helps you optimize network configuration based on your observations.</p>

AWS tool	Where to use
AWS Network Manager	Using AWS Network Manager, you can monitor the real-time and historical performance of the AWS Global Network for operational and planning purposes. Network Manager provides aggregate network latency between AWS Regions and Availability Zones and within each Availability Zone, allowing you to better understand how your application performance relates to the performance of the underlying AWS network.
Amazon CloudWatch RUM	Use Amazon CloudWatch RUM to collect the metrics that give you the insights that help you identify, understand, and improve user experience.

- Identify top talkers and application traffic patterns using VPC and AWS Transit Gateway Flow Logs.
- Assess and optimize your current network architecture including VPCs, subnets, and routing. As an example, you can evaluate how different VPC peering or AWS Transit Gateway can help you improve the networking in your architecture.
- Assess the routing paths in your network to verify that the shortest path between destinations is always used. Network Access Analyzer can help you do this.

Resources

Related documents:

- [VPC Flow Logs](#)
- [Public DNS query logging](#)
- [What is IPAM?](#)
- [What is Reachability Analyzer?](#)
- [What is Network Access Analyzer?](#)
- [CloudWatch metrics for your VPCs](#)

- [Optimize performance and reduce costs for network analytics with VPC Flow Logs in Apache Parquet format](#)
- [Monitoring your global and core networks with Amazon CloudWatch metrics](#)
- [Continuously monitor network traffic and resources](#)

Related videos:

- [Networking best practices and tips with the AWS Well-Architected Framework](#)
- [Monitoring and troubleshooting network traffic](#)

Related examples:

- [AWS Networking Workshops](#)
- [AWS Network Monitoring](#)

Process and culture

Questions

- [PERF 5. How do your organizational practices and culture contribute to performance efficiency in your workload?](#)

PERF 5. How do your organizational practices and culture contribute to performance efficiency in your workload?

When architecting workloads, there are principles and practices that you can adopt to help you better run efficient high-performing cloud workloads. To adopt a culture that fosters performance efficiency of cloud workloads, consider these key principles and practices:

Best practices

- [PERF05-BP01 Establish key performance indicators \(KPIs\) to measure workload health and performance](#)
- [PERF05-BP02 Use monitoring solutions to understand the areas where performance is most critical](#)
- [PERF05-BP03 Define a process to improve workload performance](#)

- [PERF05-BP04 Load test your workload](#)
- [PERF05-BP05 Use automation to proactively remediate performance-related issues](#)
- [PERF05-BP06 Keep your workload and services up-to-date](#)
- [PERF05-BP07 Review metrics at regular intervals](#)

PERF05-BP01 Establish key performance indicators (KPIs) to measure workload health and performance

Identify the KPIs that quantitatively and qualitatively measure workload performance. KPIs help you measure the health and performance of a workload related to a business goal.

Common anti-patterns:

- You only monitor system-level metrics to gain insight into your workload and don't understand business impacts to those metrics.
- You assume that your KPIs are already being published and shared as standard metric data.
- You do not define a quantitative, measurable KPI.
- You do not align KPIs with business goals or strategies.

Benefits of establishing this best practice: Identifying specific KPIs that represent workload health and performance helps align teams on their priorities and define successful business outcomes. Sharing those metrics with all departments provides visibility and alignment on thresholds, expectations, and business impact.

Level of risk exposed if this best practice is not established: High

Implementation guidance

KPIs allow business and engineering teams to align on the measurement of goals and strategies and how these factors combine to produce business outcomes. For example, a website workload might use page load time as an indication of overall performance. This metric would be one of multiple data points that measures user experience. In addition to identifying the page load time thresholds, you should document the expected outcome or business risk if ideal performance is not met. A long page load time affects your end users directly, decreases their user experience rating, and can lead to a loss of customers. When you define your KPI thresholds, combine both industry benchmarks and your end user expectations. For example, if the current industry benchmark is a webpage loading within a two-second time period, but your end users expect a webpage to load

within a one-second time period, then you should take both of these data points into consideration when establishing the KPI.

Your team must evaluate your workload KPIs using real-time granular data and historical data for reference and create dashboards that perform metric math on your KPI data to derive operational and utilization insights. KPIs should be documented and include thresholds that support business goals and strategies, and should be mapped to metrics being monitored. KPIs should be revisited when business goals, strategies, or end user requirements change.

Implementation steps

1. Identify and document key business stakeholders.
2. Work with these stakeholders to define and document objectives of your workload.
3. Review industry best practices to identify relevant KPIs aligned with your workload objectives.
4. Use industry best practices and your workload objectives to set targets for your workload KPI.
 Use this information to set KPI thresholds for severity or alarm level.
5. Identify and document the risk and impact if the KPI is not met.
6. Identify and document metrics that can help you to establish the KPIs.
7. Use monitoring tools such as [Amazon CloudWatch](#) or [AWS Config](#) to collect metrics and measure KPIs.
8. Use dashboards to visualize and communicate KPIs with stakeholders.
9. Regularly review and analyze metrics to identify areas of workload that need to be improved.
10. Revisit KPIs when business goals or workload performance changes.

Resources

Related documents:

- [CloudWatch documentation](#)
- [Monitoring, Logging, and Performance AWS Partners](#)
- [X-Ray Documentation](#)
- [Using Amazon CloudWatch dashboards](#)
- [Amazon QuickSight KPIs](#)

Related videos:

- [AWS re:Invent 2019: Scaling up to your first 10 million users](#)
- [Cut through the chaos: Gain operational visibility and insight](#)
- [Build a Monitoring Plan](#)

Related examples:

- [Creating a dashboard with Amazon QuickSight](#)

PERF05-BP02 Use monitoring solutions to understand the areas where performance is most critical

Understand and identify areas where increasing the performance of your workload will have a positive impact on efficiency or customer experience. For example, a website that has a large amount of customer interaction can benefit from using edge services to move content delivery closer to customers.

Common anti-patterns:

- You assume that standard compute metrics such as CPU utilization or memory pressure are enough to catch performance issues.
- You only use the default metrics recorded by your selected monitoring software.
- You only review metrics when there is an issue.

Benefits of establishing this best practice: Understanding critical areas of performance helps workload owners monitor KPIs and prioritize high-impact improvements.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Set up end-to-end tracing to identify traffic patterns, latency, and critical performance areas. Monitor your data access patterns for slow queries or poorly fragmented and partitioned data. Identify the constrained areas of the workload using load testing or monitoring.

Increase performance efficiency by understanding your architecture, traffic patterns, and data access patterns, and identify your latency and processing times. Identify the potential bottlenecks that might affect the customer experience as the workload grows. After investigating these areas, look at which solution you could deploy to remove those performance concerns.

Implementation steps

- Set up end-to-end monitoring to capture all workload components and metrics. Here are examples of monitoring solutions on AWS.

Service	Where to use
Amazon CloudWatch Real-User Monitoring (RUM)	To capture application performance metrics from real user client-side and frontend sessions.
AWS X-Ray	To trace traffic through the application layers and identify latency between components and dependencies. Use X-Ray service maps to see relationships and latency between workload components.
Amazon Relational Database Service Performance Insights	To view database performance metrics and identify performance improvements.
Amazon RDS Enhanced Monitoring	To view database OS performance metrics.
Amazon DevOps Guru	To detect abnormal operating patterns so you can identify operational issues before they impact your customers.

- Perform tests to generate metrics, identify traffic patterns, bottlenecks, and critical performance areas. Here are some examples of how to perform testing:

- Set up [CloudWatch Synthetic Canaries](#) to mimic browser-based user activities programmatically using Linux cron jobs or rate expressions to generate consistent metrics over time.
- Use the [AWS Distributed Load Testing](#) solution to generate peak traffic or test the workload at the expected growth rate.

- Evaluate the metrics and telemetry to identify your critical performance areas. Review these areas with your team to discuss monitoring and solutions to avoid bottlenecks.

- Experiment with performance improvements and measure those changes with data. As an example, you can use [CloudWatch Evidently](#) to test new improvements and performance impacts to your workload.

Resources

Related documents:

- [Amazon Builders' Library](#)
- [X-Ray Documentation](#)
- [Amazon CloudWatch RUM](#)
- [Amazon DevOps Guru](#)

Related videos:

- [The Amazon Builders' Library: 25 years of Amazon operational excellence](#)
- [Visual Monitoring of Applications with Amazon CloudWatch Synthetics](#)

Related examples:

- [Measure page load time with Amazon CloudWatch Synthetics](#)
- [Amazon CloudWatch RUM Web Client](#)
- [X-Ray SDK for Node.js](#)
- [X-Ray SDK for Python](#)
- [X-Ray SDK for Java](#)
- [X-Ray SDK for .Net](#)
- [X-Ray SDK for Ruby](#)
- [X-Ray Daemon](#)
- [Distributed Load Testing on AWS](#)

PERF05-BP03 Define a process to improve workload performance

Define a process to evaluate new services, design patterns, resource types, and configurations as they become available. For example, run existing performance tests on new instance offerings to determine their potential to improve your workload.

Common anti-patterns:

- You assume your current architecture is static and won't be updated over time.
- You introduce architecture changes over time with no metric justification.

Benefits of establishing this best practice: By defining your process for making architectural changes, you can use gathered data to influence your workload design over time.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Your workload's performance has a few key constraints. Document these so that you know what kinds of innovation might improve the performance of your workload. Use this information when learning about new services or technology as it becomes available to identify ways to alleviate constraints or bottlenecks.

Identify the key performance constraints for your workload. Document your workload's performance constraints so that you know what kinds of innovation might improve the performance of your workload.

Implementation steps

- Identify your workload performance KPIs as outlined in [PERF05-BP01 Establish key performance indicators \(KPIs\) to measure workload health and performance](#) to baseline your workload.
- Use [AWS observability tools](#) to collect performance metrics and measure KPIs.
- Conduct in-depth analysis to identify the areas (like configuration and application code) in your workload that is under-performing as outlined in [PERF05-BP02 Use monitoring solutions to understand the areas where performance is most critical](#).
- Use your analysis and performance tools to identify the performance optimization strategy.
- Use sandbox or pre-production environments to validate effectiveness of the strategy.
- Implement the changes in production and continually monitor the workload's performance.
- Document the improvements and communicate that to stakeholders.

Resources

Related documents:

- [AWS Blog](#)
- [What's New with AWS](#)

Related videos:

- [AWS Events YouTube Channel](#)
- [AWS Online Tech Talks YouTube Channel](#)
- [Amazon Web Services YouTube Channel](#)

Related examples:

- [AWS Github](#)
- [AWS Skill Builder](#)

PERF05-BP04 Load test your workload

Load test your workload to verify it can handle production load and identify any performance bottleneck.

Common anti-patterns:

- You load test individual parts of your workload but not your entire workload.
- You load test on infrastructure that is not the same as your production environment.
- You only conduct load testing to your expected load and not beyond, to help foresee where you may have future problems.
- You perform load testing without consulting the [Amazon EC2 Testing Policy](#) and submitting a Simulated Event Submissions Form. This results in your test failing to run, as it looks like a denial-of-service event.

Benefits of establishing this best practice: Measuring your performance under a load test will show you where you will be impacted as load increases. This can provide you with the capability of anticipating needed changes before they impact your workload.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Load testing in the cloud is a process to measure the performance of cloud workload under realistic conditions with expected user load. This process involves provisioning a production-like cloud environment, using load testing tools to generate load, and analyzing metrics to assess the ability of your workload handling a realistic load. Load tests must be run using synthetic or sanitized

versions of production data (remove sensitive or identifying information). Automatically carry out load tests as part of your delivery pipeline, and compare the results against pre-defined KPIs and thresholds. This process helps you continue to achieve required performance.

Implementation steps

- Set up the test environment based on your production environment. You can use AWS services to run production-scale environments to test your architecture.
- Choose and configure the load testing tool that suits your workload.
- Define the load testing scenarios and parameters (like test duration and number of users).
- Perform test scenarios at scale. Take advantage of the AWS Cloud to test your workload to discover where it fails to scale, or if it scales in a non-linear way. For example, use Spot Instances to generate loads at low cost and discover bottlenecks before they are experienced in production.
- Monitor and record performance metrics (like throughput and response time). Amazon CloudWatch can collect metrics across the resources in your architecture. You can also collect and publish custom metrics to surface business or derived metrics.
- Analyze the results to identify performance bottlenecks and areas for improvements.
- Document and report on load testing process and results.

Resources

Related documents:

- [AWS CloudFormation](#)
- [Amazon CloudWatch RUM](#)
- [Amazon CloudWatch Synthetics](#)
- [Distributed Load Testing on AWS](#)

Related videos:

- [Solving with AWS Solutions: Distributed Load Testing](#)
- [Optimize applications through Amazon CloudWatch RUM](#)
- [Demo of Amazon CloudWatch Synthetics](#)

Related examples:

- [Distributed Load Testing on AWS](#)

PERF05-BP05 Use automation to proactively remediate performance-related issues

Use key performance indicators (KPIs), combined with monitoring and alerting systems, to proactively address performance-related issues.

Common anti-patterns:

- You only allow operations staff the ability to make operational changes to the workload.
- You let all alarms filter to the operations team with no proactive remediation.

Benefits of establishing this best practice: Proactive remediation of alarm actions allows support staff to concentrate on those items that are not automatically actionable. This helps operations staff handle all alarms without being overwhelmed and instead focus only on critical alarms.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Use alarms to trigger automated actions to remediate issues where possible. Escalate the alarm to those able to respond if automated response is not possible. For example, you may have a system that can predict expected key performance indicator (KPI) values and alarm when they breach certain thresholds, or a tool that can automatically halt or roll back deployments if KPIs are outside of expected values.

Implement processes that provide visibility into performance as your workload is running. Build monitoring dashboards and establish baseline norms for performance expectations to determine if the workload is performing optimally.

Implementation steps

- Identify and understand the performance issue that can be remediated automatically. Use AWS monitoring solutions such as [Amazon CloudWatch](#) or AWS X-Ray to help you better understand the root cause of the issue.
- Create a step-by-step remediation plan and process that can be used to automatically fix the issue.

- Configure the trigger to automatically initiate the remediation process. For example, you can define a trigger to automatically restart an instance when it reaches a certain threshold of CPU utilization.
- Use AWS services and technologies to automate the remediation process. For example, [AWS Systems Manager Automation](#) provides a secure and scalable way to automate the remediation process.
- Test the automated remediation process in a pre-production environment.
- After testing, implement the remediation process in production environment and continually monitor to identify areas for improvement.

Resources

Related documents:

- [CloudWatch Documentation](#)
- [Monitoring, Logging, and Performance AWS Partner Network Partners](#)
- [X-Ray Documentation](#)
- [Using Alarms and Alarm Actions in CloudWatch](#)

Related videos:

- [Intelligently automating cloud operations](#)
- [Setting up controls at scale in your AWS environment](#)
- [Automating patch management and compliance using AWS](#)
- [How Amazon uses better metrics for improved website performance](#)

Related examples:

- [CloudWatch Logs Customize Alarms](#)

PERF05-BP06 Keep your workload and services up-to-date

Stay up-to-date on new cloud services and features to adopt efficient features, remove issues, and improve the overall performance efficiency of your workload.

Common anti-patterns:

- You assume your current architecture is static and will not be updated over time.
- You do not have any systems or a regular cadence to evaluate if updated software and packages are compatible with your workload.

Benefits of establishing this best practice: By establishing a process to stay up-to-date on new services and offerings, you can adopt new features and capabilities, resolve issues, and improve workload performance.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Evaluate ways to improve performance as new services, design patterns, and product features become available. Determine which of these could improve performance or increase the efficiency of the workload through evaluation, internal discussion, or external analysis. Define a process to evaluate updates, new features, and services relevant to your workload. For example, build a proof of concept that uses new technologies or consult with an internal group. When trying new ideas or services, run performance tests to measure the impact that they have on the performance of the workload.

Implementation steps

- Inventory your workload software and architecture and identify components that need to be updated.
- Identify news and update sources related to your workload components. As an example, you can subscribe to the [What's New at AWS blog](#) for the products that match your workload component. You can subscribe to the RSS feed or manage your [email subscriptions](#).
- Define a schedule to evaluate new services and features for your workload.
 - You can use [AWS Systems Manager Inventory](#) to collect operating system (OS), application, and instance metadata from your Amazon EC2 instances and quickly understand which instances are running the software and configurations required by your software policy and which instances need to be updated.
- Understand how to update the components of your workload. Take advantage of agility in the cloud to quickly test how new features can improve your workload to gain performance efficiencies.
- Use automation for the update process to reduce the level of effort to deploy new features and limit errors caused by manual processes.

- You can use [CI/CD](#) to automatically update AMIs, container images, and other artifacts related to your cloud application.
- You can use tools such as [AWS Systems Manager Patch Manager](#) to automate the process of system updates, and schedule the activity using [AWS Systems Manager Maintenance Windows](#).
- Document your process for evaluating updates and new services. Provide your owners the time and space needed to research, test, experiment, and validate updates and new services. Refer back to the documented business requirements and KPIs to help prioritize which update will make a positive business impact.

Resources

Related documents:

- [AWS Blog](#)
- [What's New with AWS](#)

Related videos:

- [AWS Events YouTube Channel](#)
- [AWS Online Tech Talks YouTube Channel](#)
- [Amazon Web Services YouTube Channel](#)

Related examples:

- [Well-Architected Labs - Inventory and Patch Management](#)
- [Lab: AWS Systems Manager](#)

PERF05-BP07 Review metrics at regular intervals

As part of routine maintenance or in response to events or incidents, review which metrics are collected. Use these reviews to identify which metrics were essential in addressing issues and which additional metrics, if they were being tracked, could help identify, address, or prevent issues.

Common anti-patterns:

- You allow metrics to stay in an alarm state for an extended period of time.

- You create alarms that are not actionable by an automation system.

Benefits of establishing this best practice: Continually review metrics that are being collected to verify that they properly identify, address, or prevent issues. Metrics can also become stale if you let them stay in an alarm state for an extended period of time.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Constantly improve metric collection and monitoring. As part of responding to incidents or events, evaluate which metrics were helpful in addressing the issue and which metrics could have helped that are not currently being tracked. Use this method to improve the quality of metrics you collect so that you can prevent, or more quickly resolve future incidents.

As part of responding to incidents or events, evaluate which metrics were helpful in addressing the issue and which metrics could have helped that are not currently being tracked. Use this to improve the quality of metrics you collect so that you can prevent or more quickly resolve future incidents.

Implementation steps

1. Define critical performance metrics to monitor that are aligned to your workload objective.
2. Set a baseline and desirable value for each metric.
3. Set a cadence (like weekly or monthly) to review critical metrics.
4. During each review, assess trends and deviation from the baseline values. Look for any performance bottlenecks or anomalies.
5. For identified issues, conduct in-depth root cause analysis to understand the main reason behind the issue.
6. Document your findings and use strategies to deal with identified issues and bottlenecks.
7. Continually assess and improve the metrics review process.

Resources

Related documents:

- [CloudWatch Documentation](#)
- [Collect metrics and logs from Amazon EC2 Instances and on-premises servers with the CloudWatch Agent](#)

- [Monitoring, Logging, and Performance AWS Partner Network Partners](#)
- [X-Ray Documentation](#)

Related videos:

- [Setting up controls at scale in your AWS environment](#)
- [How Amazon uses better metrics for improved website performance](#)

Related examples:

- [Creating a dashboard with Amazon QuickSight](#)
- [Level 100: Monitoring with CloudWatch Dashboards](#)

Cost optimization

The Cost Optimization pillar includes the ability to run systems to deliver business value at the lowest price point. You can find prescriptive guidance on implementation in the [Cost Optimization Pillar whitepaper](#).

Best practice areas

- [Practice Cloud Financial Management](#)
- [Expenditure and usage awareness](#)
- [Cost-effective resources](#)
- [Manage demand and supply resources](#)
- [Optimize over time](#)

Practice Cloud Financial Management

Question

- [COST 1. How do you implement cloud financial management?](#)

COST 1. How do you implement cloud financial management?

Implementing Cloud Financial Management helps organizations realize business value and financial success as they optimize their cost and usage and scale on AWS.

Best practices

- [COST01-BP01 Establish ownership of cost optimization](#)
- [COST01-BP02 Establish a partnership between finance and technology](#)
- [COST01-BP03 Establish cloud budgets and forecasts](#)
- [COST01-BP04 Implement cost awareness in your organizational processes](#)
- [COST01-BP05 Report and notify on cost optimization](#)
- [COST01-BP06 Monitor cost proactively](#)
- [COST01-BP07 Keep up-to-date with new service releases](#)
- [COST01-BP08 Create a cost-aware culture](#)
- [COST01-BP09 Quantify business value from cost optimization](#)

COST01-BP01 Establish ownership of cost optimization

Create a team (Cloud Business Office, Cloud Center of Excellence, or FinOps team) that is responsible for establishing and maintaining cost awareness across your organization. The owner of cost optimization can be an individual or a team (requires people from finance, technology, and business teams) that understands the entire organization and cloud finance.

Level of risk exposed if this best practice is not established: High

Implementation guidance

This is the introduction of a Cloud Business Office (CBO) or Cloud Center of Excellence (CCOE) function or team that is responsible for establishing and maintaining a culture of cost awareness in cloud computing. This function can be an existing individual, a team within your organization, or a new team of key finance, technology, and organization stakeholders from across the organization.

The function (individual or team) prioritizes and spends the required percentage of their time on cost management and cost optimization activities. For a small organization, the function might spend a smaller percentage of time compared to a full-time function for a larger enterprise.

The function requires a multi-disciplinary approach, with capabilities in project management, data science, financial analysis, and software or infrastructure development. They can improve workload efficiency by running cost optimizations within three different ownerships:

- **Centralized:** Through designated teams such as FinOps team, Cloud Financial Management (CFM) team, Cloud Business Office (CBO), or Cloud Center of Excellence (CCoE), customers can design and implement governance mechanisms and drive best practices company-wide.
- **Decentralized:** Influencing technology teams to run cost optimizations.
- **Hybrid:** Combination of both centralized and decentralized teams can work together to run cost optimizations.

The function may be measured against their ability to run and deliver against cost optimization goals (for example, workload efficiency metrics).

You must secure executive sponsorship for this function, which is a key success factor. The sponsor is regarded as a champion for cost efficient cloud consumption, and provides escalation support for the team to ensure that cost optimization activities are treated with the level of priority defined by the organization. Otherwise, guidance can be ignored and cost saving opportunities will not be prioritized. Together, the sponsor and team help your organization consume the cloud efficiently and deliver business value.

If you have the Business, Enterprise-On-Ramp or Enterprise [support plan](#) and need help building this team or function, reach out to your Cloud Financial Management (CFM) experts through your account team.

Implementation steps

- **Define key members:** All relevant parts of your organization must contribute and be interested in cost management. Common teams within organizations typically include: finance, application or product owners, management, and technical teams (DevOps). Some are engaged full time (finance or technical), while others are engaged periodically as required. Individuals or teams performing CFM need the following set of skills:
 - **Software development:** in the case where scripts and automation are being built out.
 - **Infrastructure engineering:** to deploy scripts, automate processes, and understand how services or resources are provisioned.
 - **Operations acumen:** CFM is about operating on the cloud efficiently by measuring, monitoring, modifying, planning, and scaling efficient use of the cloud.

- **Define goals and metrics:** The function needs to deliver value to the organization in different ways. These goals are defined and continually evolve as the organization evolves. Common activities include: creating and running education programs on cost optimization across the organization, developing organization-wide standards, such as monitoring and reporting for cost optimization, and setting workload goals on optimization. This function also needs to regularly report to the organization on their cost optimization capability.

You can define value- or cost-based key performance indicators (KPIs). When you define the KPIs, you can calculate expected cost in terms of efficiency and expected business outcome. Value-based KPIs tie cost and usage metrics to business value drivers and help rationalize changes in AWS spend. The first step to deriving value-based KPIs is working together, cross-organizationally, to select and agree upon a standard set of KPIs.

- **Establish regular cadence:** The group (finance, technology and business teams) should come together regularly to review their goals and metrics. A typical cadence involves reviewing the state of the organization, reviewing any programs currently running, and reviewing overall financial and optimization metrics. Afterwards, key workloads are reported on in greater detail.

During these regular reviews, you can review workload efficiency (cost) and business outcome. For example, a 20% cost increase for a workload may align with increased customer usage. In this case, this 20% cost increase can be interpreted as an investment. These regular cadence calls can help teams to identify value KPIs that provide meaning to the entire organization.

Resources

Related documents:

- [AWS CCOE Blog](#)
- [Creating Cloud Business Office](#)
- [CCOE - Cloud Center of Excellence](#)

Related videos:

- [Vanguard CCOE Success Story](#)

Related examples:

- [Using a Cloud Center of Excellence \(CCOE\) to Transform the Entire Enterprise](#)

- [Building a CCOE to transform the entire enterprise](#)
- [7 Pitfalls to Avoid When Building CCOE](#)

COST01-BP02 Establish a partnership between finance and technology

Involve finance and technology teams in cost and usage discussions at all stages of your cloud journey. Teams regularly meet and discuss topics such as organizational goals and targets, current state of cost and usage, and financial and accounting practices.

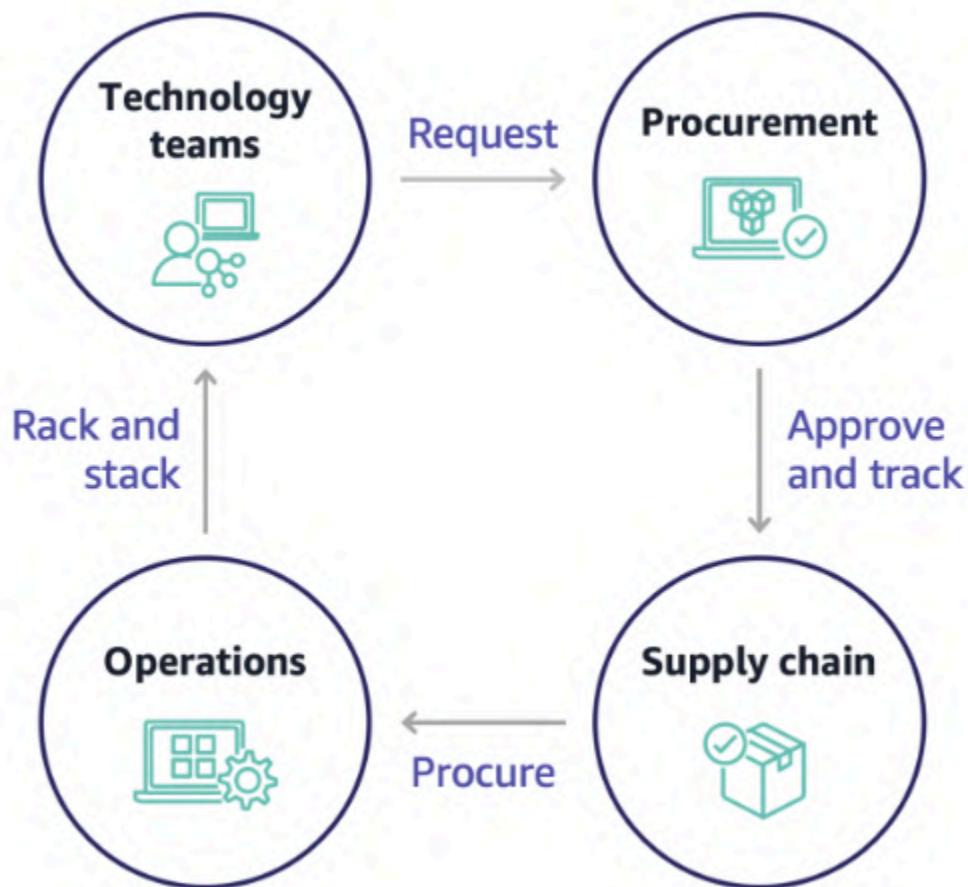
Level of risk exposed if this best practice is not established: High

Implementation guidance

Technology teams innovate faster in the cloud due to shortened approval, procurement, and infrastructure deployment cycles. This can be an adjustment for finance organizations previously used to running time-consuming and resource-intensive processes for procuring and deploying capital in data center and on-premises environments, and cost allocation only at project approval.

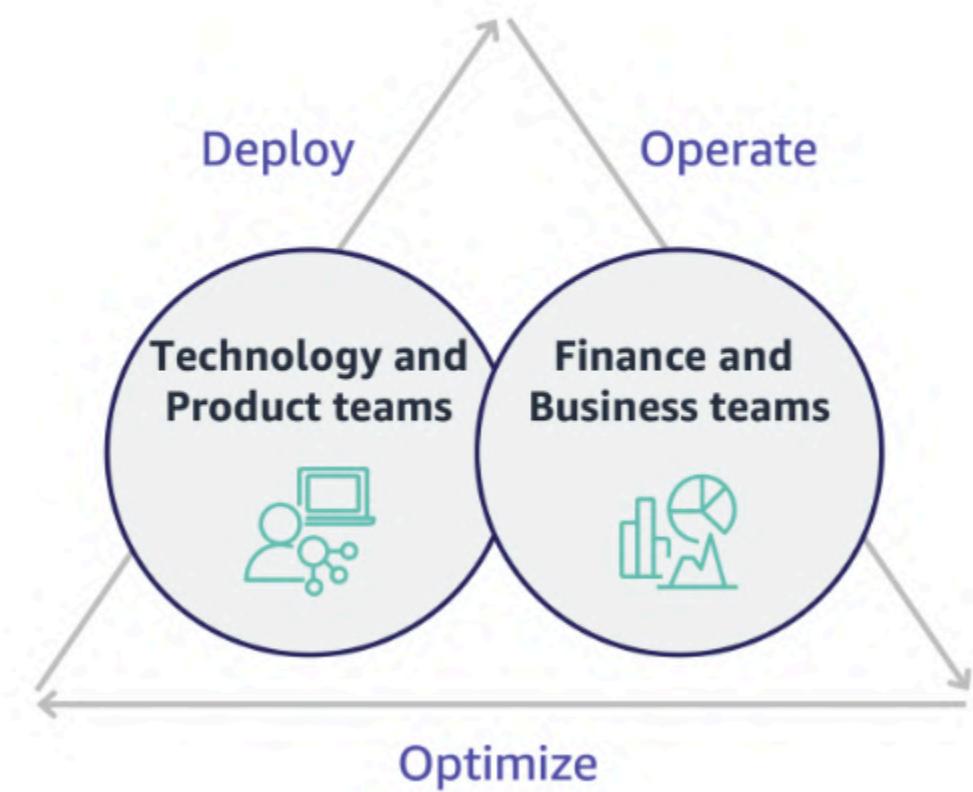
From a finance and procurement organization perspective, the process for capital budgeting, capital requests, approvals, procurement, and installing physical infrastructure is one that has been learned and standardized over decades:

- Engineering or IT teams are typically the requesters
- Various finance teams act as approvers and procurers
- Operations teams rack, stack, and hand off ready-to-use infrastructure



With the adoption of cloud, infrastructure procurement and consumption are no longer beholden to a chain of dependencies. In the cloud model, technology and product teams are no longer just builders, but operators and owners of their products, responsible for most of the activities historically associated with finance and operations teams, including procurement and deployment.

All it really takes to provision cloud resources is an account, and the right set of permissions. This is also what reduces IT and finance risk; which means teams are always a just few clicks or API calls away from terminating idle or unnecessary cloud resources. This is also what allows technology teams to innovate faster – the agility and ability to spin up and then tear down experiments. While the variable nature of cloud consumption may impact predictability from a capital budgeting and forecasting perspective, cloud provides organizations with the ability to reduce the cost of over-provisioning, as well as reduce the opportunity cost associated with conservative under-provisioning.



Establish a partnership between key finance and technology stakeholders to create a shared understanding of organizational goals and develop mechanisms to succeed financially in the variable spend model of cloud computing. Relevant teams within your organization must be involved in cost and usage discussions at all stages of your cloud journey, including:

- **Financial leads:** CFOs, financial controllers, financial planners, business analysts, procurement, sourcing, and accounts payable must understand the cloud model of consumption, purchasing options, and the monthly invoicing process. Finance needs to partner with technology teams to create and socialize an IT value story, helping business teams understand how technology spend is linked to business outcomes. This way, technology expenditures are viewed not as costs, but rather as investments. Due to the fundamental differences between the cloud (such as the rate of change in usage, pay as you go pricing, tiered pricing, pricing models, and detailed billing and usage information) compared to on-premises operation, it is essential that the finance organization understands how cloud usage can impact business aspects including procurement processes, incentive tracking, cost allocation and financial statements.
- **Technology leads:** Technology leads (including product and application owners) must be aware of the financial requirements (for example, budget constraints) as well as business requirements

(for example, service level agreements). This allows the workload to be implemented to achieve the desired goals of the organization.

The partnership of finance and technology provides the following benefits:

- Finance and technology teams have near real-time visibility into cost and usage.
- Finance and technology teams establish a standard operating procedure to handle cloud spend variance.
- Finance stakeholders act as strategic advisors with respect to how capital is used to purchase commitment discounts (for example, Reserved Instances or AWS Savings Plans), and how the cloud is used to grow the organization.
- Existing accounts payable and procurement processes are used with the cloud.
- Finance and technology teams collaborate on forecasting future AWS cost and usage to align and build organizational budgets.
- Better cross-organizational communication through a shared language, and common understanding of financial concepts.

Additional stakeholders within your organization that should be involved in cost and usage discussions include:

- **Business unit owners:** Business unit owners must understand the cloud business model so that they can provide direction to both the business units and the entire company. This cloud knowledge is critical when there is a need to forecast growth and workload usage, and when assessing longer-term purchasing options, such as Reserved Instances or Savings Plans.
- **Engineering team:** Establishing a partnership between finance and technology teams is essential for building a cost-aware culture that encourages engineers to take action on Cloud Financial Management (CFM). One of the common problems of CFM or finance operations practitioners and finance teams is getting engineers to understand the whole business on cloud, follow best practices, and take recommended actions.
- **Third parties:** If your organization uses third parties (for example, consultants or tools), ensure that they are aligned to your financial goals and can demonstrate both alignment through their engagement models and a return on investment (ROI). Typically, third parties will contribute to reporting and analysis of any workloads that they manage, and they will provide cost analysis of any workloads that they design.

Implementing CFM and achieving success requires collaboration across finance, technology, and business teams, and a shift in how cloud spend is communicated and evaluated across the organization. Include engineering teams so that they can be part of these cost and usage discussions at all stages, and encourage them to follow best practices and take agreed-upon actions accordingly.

Implementation steps

- **Define key members:** Verify that all relevant members of your finance and technology teams participate in the partnership. Relevant finance members will be those having interaction with the cloud bill. This will typically be CFOs, financial controllers, financial planners, business analysts, procurement, and sourcing. Technology members will typically be product and application owners, technical managers and representatives from all teams that build on the cloud. Other members may include business unit owners, such as marketing, that will influence usage of products, and third parties such as consultants, to achieve alignment to your goals and mechanisms, and to assist with reporting.
- **Define topics for discussion:** Define the topics that are common across the teams, or will need a shared understanding. Follow cost from that time it is created, until the bill is paid. Note any members involved, and organizational processes that are required to be applied. Understand each step or process it goes through and the associated information, such as pricing models available, tiered pricing, discount models, budgeting, and financial requirements.
- **Establish regular cadence:** To create a finance and technology partnership, establish a regular communication cadence to create and maintain alignment. The group needs to come together regularly against their goals and metrics. A typical cadence involves reviewing the state of the organization, reviewing any programs currently running, and reviewing overall financial and optimization metrics. Then key workloads are reported on in greater detail.

Resources

Related documents:

- [AWS News Blog](#)

COST01-BP03 Establish cloud budgets and forecasts

Adjust existing organizational budgeting and forecasting processes to be compatible with the highly variable nature of cloud costs and usage. Processes must be dynamic, using trend-based or business driver-based algorithms or a combination of both.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Customers use the cloud for efficiency, speed, and agility, which creates a highly variable amount of cost and usage. Costs can decrease (or sometimes increase) with increases in workload efficiency or as new workloads and features are deployed. Workloads can scale to serve more of your customers, which increases cloud usage and costs. Resources are now more readily accessible than ever before. The elasticity of the cloud also brings an elasticity of costs and forecasts. Existing organizational budgeting processes must be modified to incorporate this variability.

The budget is usually prepared for a single year and remains fixed, requiring strict adherence from everyone involved. In contrast, forecasting is more flexible, allowing for readjustments throughout the year and providing dynamic projections over a period of one, two, or three years. Both budgeting and forecasting play a crucial role in establishing financial expectations among various technology and business stakeholders. Accurate forecasting and implementation also bring accountability for stakeholders who are directly responsible for provisioning cost in the first place, and it can also raise their overall cost awareness.

Adjust existing budgeting and forecasting processes to become more dynamic using either a trend-based algorithm (using historical costs as inputs) or driver-based algorithms (for example, new product launches, Regional expansion, or new environments for workloads), which is ideal for a dynamic and variable spending environment, or a combination of both trend and business drivers.

You can use [AWS Cost Explorer](#) for trend-based forecasting in a defined future time range based on your past spend. AWS Cost Explorer's forecasting engine segments your historical data based on charge types (for example, Reserved Instances) and uses a combination of machine learning and rule-based models to predict spend across all charge types individually.

Identify the business drivers which may impact your usage cost and forecast for each of them separately to ensure expected usage is calculated in advance. Some of the drivers are linked to IT and product teams within the organization. Other business drivers, such as marketing events, promotions, mergers, and acquisitions, are known by your sales, marketing, and business leaders,

and it's important to collaborate and account for all those demand drivers as well. You need to work with them closely to understand the impact on new internal drivers.

Once you've determined your trend-based forecast using Cost Explorer or any other tools, use the [AWS Pricing Calculator](#) to estimate your AWS use case and future costs based on the expected usage (traffic, requests-per-second, or required Amazon EC2 instance). You can also use it to help you plan how you spend, find cost saving opportunities, and make informed decisions when using AWS. It is important to track the accuracy of that forecast as budgets should be set based on these forecast calculations and estimations.

Use [AWS Budgets](#) to set custom budgets at a granular level by specifying the time period, recurrence, or amount (fixed or variable), and adding filters such as service, AWS Region, and tags. To stay informed on the performance of your existing budgets, you can create and schedule [AWS Budgets Reports](#) to be emailed to you and your stakeholders on a regular cadence. You can also create [AWS Budgets Alerts](#) based on actual costs, which is reactive in nature, or on forecasted costs, which provides time to implement mitigations against potential cost overruns. You can be alerted when your cost or usage exceeds, or if they are forecasted to exceed, your budgeted amount.

Use [AWS Cost Anomaly Detection](#) to prevent or reduce cost surprises and enhance control without slowing innovation. AWS Cost Anomaly Detection leverages machine learning to identify anomalous spend and root causes, so you can quickly take action. [With three simple steps](#), you can create your own contextualized monitor and receive alerts when any anomalous spend is detected.

As mentioned in the Well-Architected cost optimization pillar's [Finance and Technology Partnership section](#), it is important to have partnership and cadences between IT, finance, and other stakeholders to verify that they are all using the same tools or processes for consistency. In cases where budgets may need to change, increasing cadence touch points can help react to those changes more quickly.

Implementation steps

- **Analyze trend-based forecasting:** Use preferred trend-based forecasting tools such as AWS Cost Explorer and Amazon Forecast. Analyze your usage cost on the different dimensions like service, account, tags and cost categories. If advanced forecasting is required, import your AWS Cost and Usage Report data into Amazon Forecast (which applies linear regression as a form of machine learning to forecast).
- **Analyze driver-based forecasting:** Identify the impact of business drivers on your cloud usage and forecast for each of them separately to calculate expected usage cost in advance. Work

closely with business unit owners and stakeholders to understand the impact on new drivers and calculate expected cost changes to define accurate budgets.

- **Update existing forecasting and budget processes:** Define your forecasting budgeting processes based on adopted forecasting methods such as trend-based, business driver-based, or a combination of both forecasting methods. Budgets should be calculated and realistic, based on these forecasting processes.
- **Configure alerts and notifications:** Use AWS Budgets Alerts and AWS Cost Anomaly Detection to get alerts and notifications.
- **Perform regular reviews with key stakeholders:** For example, stakeholders in IT, finance, platform teams, and other areas of the business, should align with changes in business direction and usage.

Resources

Related documents:

- [AWS Cost Explorer](#)
- [AWS Cost and Usage Report](#)
- [Amazon QuickSight Forecasting](#)
- [Amazon Forecast](#)
- [AWS Budgets](#)
- [AWS News Blog](#)

Related videos:

- [How can I use AWS Budgets to track my spending and usage](#)
- [AWS Cost Optimization Series: AWS Budgets](#)

Related examples:

- [Understand and build driver-based forecasting](#)
- [How to establish and drive a forecasting culture](#)
- [How to improve your cloud cost forecasting](#)
- [Using the right tools for your cloud cost forecasting](#)

COST01-BP04 Implement cost awareness in your organizational processes

Implement cost awareness, create transparency, and accountability of costs into new or existing processes that impact usage, and leverage existing processes for cost awareness. Implement cost awareness into employee training.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Cost awareness must be implemented in new and existing organizational processes. It is one of the foundational, prerequisite capabilities for other best practices. It is recommended to reuse and modify existing processes where possible — this minimizes the impact to agility and velocity. Report cloud costs to the technology teams and the decision makers in the business and finance teams to raise cost awareness, and establish efficiency key performance indicators (KPIs) for finance and business stakeholders. The following recommendations will help implement cost awareness in your workload:

- Verify that change management includes a cost measurement to quantify the financial impact of your changes. This helps proactively address cost-related concerns and highlight cost savings.
- Verify that cost optimization is a core component of your operating capabilities. For example, you can leverage existing incident management processes to investigate and identify root causes for cost and usage anomalies or cost overruns.
- Accelerate cost savings and business value realization through automation or tooling. When thinking about the cost of implementing, frame the conversation to include an return on investment (ROI) component to justify the investment of time or money.
- Allocate cloud costs by implementing showbacks or chargebacks for cloud spend, including spend on commitment-based purchase options, shared services and marketplace purchases to drive most cost-aware cloud consumption.
- Extend existing training and development programs to include cost-awareness training throughout your organization. It is recommended that this includes continuous training and certification. This will build an organization that is capable of self-managing cost and usage.
- Take advantage of free AWS native tools such as [AWS Cost Anomaly Detection](#), [AWS Budgets](#), and [AWS Budgets Reports](#).

When organizations consistently adopt [Cloud Financial Management](#) (CFM) practices, those behaviours become ingrained in the way of working and decision-making. The result is a culture

that is more cost-aware, from developers architecting a new born-in-the-cloud application, to finance managers analyzing the ROI on these new cloud investments.

Implementation steps

- **Identify relevant organizational processes:** Each organizational unit reviews their processes and identifies processes that impact cost and usage. Any processes that result in the creation or termination of a resource need to be included for review. Look for processes that can support cost awareness in your business, such as incident management and training.
- **Establish self-sustaining cost-aware culture:** Make sure all the relevant stakeholders align with cause-of-change and impact as a cost so that they understand cloud cost. This will allow your organization to establish a self-sustaining cost-aware culture of innovation.
- **Update processes with cost awareness:** Each process is modified to be made cost aware. The process may require additional pre-checks, such as assessing the impact of cost, or post-checks validating that the expected changes in cost and usage occurred. Supporting processes such as training and incident management can be extended to include items for cost and usage.

To get help, reach out to CFM experts through your Account team, or explore the resources and related documents below.

Resources

Related documents:

- [AWS Cloud Financial Management](#)

Related examples:

- [Strategy for Efficient Cloud Cost Management](#)
- [Cost Control Blog Series #3: How to Handle Cost Shock](#)
- [A Beginner's Guide to AWS Cost Management](#)

COST01-BP05 Report and notify on cost optimization

Set up cloud budgets and configure mechanisms to detect anomalies in usage. Configure related tools for cost and usage alerts against pre-defined targets and receive notifications when any usage exceeds those targets. Have regular meetings to analyze the cost-effectiveness of your workloads and promote cost awareness.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

You must regularly report on cost and usage optimization within your organization. You can implement dedicated sessions to discuss cost performance, or include cost optimization in your regular operational reporting cycles for your workloads. Use services and tools to monitor your cost performances regularly and implement cost savings opportunities.

View your cost and usage with multiple filters and granularity by using [AWS Cost Explorer](#), which provides dashboards and reports such as costs by service or by account, daily costs, or marketplace costs. Track your progress of cost and usage against configured budgets with [AWS Budgets Reports](#).

Use [AWS Budgets](#) to set custom budgets to track your costs and usage and respond quickly to alerts received from email or Amazon Simple Notification Service (Amazon SNS) notifications if you exceed your threshold. [Set your preferred budget](#) period to daily, monthly, quarterly, or annually, and create specific budget limits to stay informed on how actual or forecasted costs and usage progress toward your budget threshold. You can also set up [alerts](#) and [actions](#) against those alerts to run automatically or through an approval process when a budget target is exceeded.

Implement notifications on cost and usage to ensure that changes in cost and usage can be acted upon quickly if they are unexpected. [AWS Cost Anomaly Detection](#) allows you to reduce cost surprises and enhance control without slowing innovation. AWS Cost Anomaly Detection identifies anomalous spend and root causes, which helps to reduce the risk of billing surprises. With three simple steps, you can create your own contextualized monitor and receive alerts when any anomalous spend is detected.

You can also use [Amazon QuickSight](#) with AWS Cost and Usage Report (CUR) data, to provide highly customized reporting with more granular data. Amazon QuickSight allows you to schedule reports and receive periodic Cost Report emails for historical cost and usage or cost-saving opportunities. Check our [Cost Intelligence Dashboard](#) (CID) solution built on Amazon QuickSight, which gives you advanced visibility.

Use [AWS Trusted Advisor](#), which provides guidance to verify whether provisioned resources are aligned with AWS best practices for cost optimization.

Check your Savings Plans recommendations through visual graphs against your granular cost and usage. Hourly graphs show On-Demand spend alongside the recommended Savings Plans

commitment, providing insight into estimated savings, Savings Plans coverage, and Savings Plans utilization. This helps organizations to understand how their Savings Plans apply to each hour of spend without having to invest time and resources into building models to analyze their spend.

Periodically create reports containing a highlight of Savings Plans, Reserved Instances, and Amazon EC2 rightsizing recommendations from AWS Cost Explorer to start reducing the cost associated with steady-state workloads, idle, and underutilized resources. Identify and recoup spend associated with cloud waste for resources that are deployed. Cloud waste occurs when incorrectly-sized resources are created or different usage patterns are observed instead what is expected. Follow AWS best practices to reduce your waste or ask your account team and partner to help you to [optimize and save](#) your cloud costs.

Generate reports regularly for better purchasing options for your resources to drive down unit costs for your workloads. Purchasing options such as Savings Plans, Reserved Instances, or Amazon EC2 Spot Instances offer the deepest cost savings for fault-tolerant workloads and allow stakeholders (business owners, finance, and tech teams) to be part of these commitment discussions.

Share the reports that contain opportunities or new release announcements that may help you to reduce total cost of ownership (TCO) of the cloud. Adopt new services, Regions, features, solutions, or new ways to achieve further cost reductions.

Implementation steps

- **Configure AWS Budgets:** Configure AWS Budgets on all accounts for your workload. Set a budget for the overall account spend, and a budget for the workload by using tags.
 - [Well-Architected Labs: Cost and Governance Usage](#)
- **Report on cost optimization:** Set up a regular cycle to discuss and analyze the efficiency of the workload. Using the metrics established, report on the metrics achieved and the cost of achieving them. Identify and fix any negative trends, as well as positive trends that you can promote across your organization. Reporting should involve representatives from the application teams and owners, finance, and key decision makers with respect to cloud expenditure.

Resources

Related documents:

- [AWS Cost Explorer](#)

- [AWS Trusted Advisor](#)
- [AWS Budgets](#)
- [AWS Cost and Usage Report](#)
- [AWS Budgets Best Practices](#)
- [Amazon S3 Analytics](#)

Related examples:

- [Well-Architected Labs: Cost and Governance Usage](#)
- [Key ways to start optimizing your AWS cloud costs](#)

COST01-BP06 Monitor cost proactively

Implement tooling and dashboards to monitor cost proactively for the workload. Regularly review the costs with configured tools or out of the box tools, do not just look at costs and categories when you receive notifications. Monitoring and analyzing costs proactively helps to identify positive trends and allows you to promote them throughout your organization.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

It is recommended to monitor cost and usage proactively within your organization, not just when there are exceptions or anomalies. Highly visible dashboards throughout your office or work environment ensure that key people have access to the information they need, and indicate the organization's focus on cost optimization. Visible dashboards allow you to actively promote successful outcomes and implement them throughout your organization.

Create a daily or frequent routine to use [AWS Cost Explorer](#) or any other dashboard such as [Amazon QuickSight](#) to see the costs and analyze proactively. Analyze AWS service usage and costs at the AWS account-level, workload-level, or specific AWS service-level with grouping and filtering, and validate whether they are expected or not. Use the hourly- and resource-level granularity and tags to filter and identify incurring costs for the top resources. You can also build your own reports with the [Cost Intelligence Dashboard](#), an [Amazon QuickSight](#) solution built by AWS Solutions Architects, and compare your budgets with the actual cost and usage.

Implementation steps

- **Report on cost optimization:** Set up a regular cycle to discuss and analyze the efficiency of the workload. Using the metrics established, report on the metrics achieved and the cost of achieving them. Identify and fix any negative trends, and identify positive trends to promote across your organization. Reporting should involve representatives from the application teams and owners, finance, and management.
- **Create and activate daily granularity [AWS Budgets](#) for the cost and usage to take timely actions to prevent any potential cost overruns:** AWS Budgets allow you to configure alert notifications, so you stay informed if any of your budget types fall out of your pre-configured thresholds. The best way to leverage AWS Budgets is to set your expected cost and usage as your limits, so that anything above your budgets can be considered overspend.
- **Create AWS Cost Anomaly Detection for cost monitor:** [AWS Cost Anomaly Detection](#) uses advanced Machine Learning technology to identify anomalous spend and root causes, so you can quickly take action. It allows you to configure cost monitors that define spend segments you want to evaluate (for example, individual AWS services, member accounts, cost allocation tags, and cost categories), and lets you set when, where, and how you receive your alert notifications. For each monitor, attach multiple alert subscriptions for business owners and technology teams, including a name, a cost impact threshold, and alerting frequency (individual alerts, daily summary, weekly summary) for each subscription.
- **Use AWS Cost Explorer or integrate your AWS Cost and Usage Report (CUR) data with Amazon QuickSight dashboards to visualize your organization's costs:** AWS Cost Explorer has an easy-to-use interface that lets you visualize, understand, and manage your AWS costs and usage over time. The [Cost Intelligence Dashboard](#) is a customizable and accessible dashboard to help create the foundation of your own cost management and optimization tool.

Resources

Related documents:

- [AWS Budgets](#)
- [AWS Cost Explorer](#)
- [Daily Cost and Usage Budgets](#)
- [AWS Cost Anomaly Detection](#)

Related examples:

- [Well-Architected Labs: Visualization](#)

- [Well-Architected Labs: Advanced Visualization](#)
- [Well-Architected Labs: Cloud Intelligence Dashboards](#)
- [Well-Architected Labs: Cost Visualization](#)
- [AWS Cost Anomaly Detection Alert with Slack](#)

COST01-BP07 Keep up-to-date with new service releases

Consult regularly with experts or AWS Partners to consider which services and features provide lower cost. Review AWS blogs and other information sources.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

AWS is constantly adding new capabilities so you can leverage the latest technologies to experiment and innovate more quickly. You may be able to implement new AWS services and features to increase cost efficiency in your workload. Regularly review [AWS Cost Management](#), the [AWS News Blog](#), the [AWS Cost Management blog](#), and [What's New with AWS](#) for information on new service and feature releases. What's New posts provide a brief overview of all AWS service, feature, and Region expansion announcements as they are released.

Implementation steps

- **Subscribe to blogs:** Go to the AWS blogs pages and subscribe to the What's New Blog and other relevant blogs. You can sign up on the [communication preference](#) page with your email address.
- **Subscribe to AWS News:** Regularly review the [AWS News Blog](#) and [What's New with AWS](#) for information on new service and feature releases. Subscribe to the RSS feed, or with your email to follow announcements and releases.
- **Follow AWS Price Reductions:** Regular price cuts on all our services has been a standard way for AWS to pass on the economic efficiencies to our customers gained from our scale. As of 2024, AWS has reduced prices 115 times since it was launched in 2006. If you have any pending business decisions due to price concerns, you can review them again after price reductions and new service integrations. You can learn about the previous price reductions efforts, including Amazon Elastic Compute Cloud (Amazon EC2) instances, in the [price-reduction category of the AWS News Blog](#).
- **AWS events and meetups:** Attend your local AWS summit, and any local meetups with other organizations from your local area. If you cannot attend in person, try to attend virtual events to hear more from AWS experts and other customers' business cases.

- **Meet with your account team:** Schedule a regular cadence with your account team, meet with them and discuss industry trends and AWS services. Speak with your account manager, Solutions Architect, and support team.

Resources

Related documents:

- [AWS Cost Management](#)
- [What's New with AWS](#)
- [AWS News Blog](#)

Related examples:

- [Amazon EC2 – 15 Years of Optimizing and Saving Your IT Costs](#)
- [AWS News Blog - Price Reduction](#)

COST01-BP08 Create a cost-aware culture

Implement changes or programs across your organization to create a cost-aware culture. It is recommended to start small, then as your capabilities increase and your organization's use of the cloud increases, implement large and wide ranging programs.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

A cost-aware culture allows you to scale cost optimization and Cloud Financial Management (financial operations, cloud center of excellence, cloud operations teams, and so on) through best practices that are performed in an organic and decentralized manner across your organization. Cost awareness allows you to create high levels of capability across your organization with minimal effort, compared to a strict top-down, centralized approach.

Creating cost awareness in cloud computing, especially for primary cost drivers in cloud computing, allows teams to understand expected outcomes of any changes in cost perspective. Teams who access the cloud environments should be aware of pricing models and the difference between traditional on-premises datacenters and cloud computing.

The main benefit of a cost-aware culture is that technology teams optimize costs proactively and continually (for example, they are considered a non-functional requirement when architecting new workloads, or making changes to existing workloads) rather than performing reactive cost optimizations as needed.

Small changes in culture can have large impacts on the efficiency of your current and future workloads. Examples of this include:

- Giving visibility and creating awareness in engineering teams to understand what they do, and what they impact in terms of cost.
- Gamifying cost and usage across your organization. This can be done through a publicly visible dashboard, or a report that compares normalized costs and usage across teams (for example, cost-per-workload and cost-per-transaction).
- Recognizing cost efficiency. Reward voluntary or unsolicited cost optimization accomplishments publicly or privately, and learn from mistakes to avoid repeating them in the future.
- Creating top-down organizational requirements for workloads to run at pre-defined budgets.
- Questioning business requirements of changes, and the cost impact of requested changes to the architecture infrastructure or workload configuration to make sure you pay only what you need.
- Making sure the change planner is aware of expected changes that have a cost impact, and that they are confirmed by the stakeholders to deliver business outcomes cost-effectively.

Implementation steps

- **Report cloud costs to technology teams:** To raise cost awareness, and establish efficiency KPIs for finance and business stakeholders.
- **Inform stakeholders or team members about planned changes:** Create an agenda item to discuss planned changes and the cost-benefit impact on the workload during weekly change meetings.
- **Meet with your account team:** Establish a regular meeting cadence with your account team, and discuss industry trends and AWS services. Speak with your account manager, architect, and support team.
- **Share success stories:** Share success stories about cost reduction for any workload, AWS account, or organization to create a positive attitude and encouragement around cost optimization.
- **Training:** Ensure technical teams or team members are trained for awareness of resource costs on AWS Cloud.

- **AWS events and meetups:** Attend local AWS summits, and any local meetups with other organizations from your local area.
- **Subscribe to blogs:** Go to the AWS blogs pages and subscribe to the [What's New Blog](#) and other relevant blogs to follow new releases, implementations, examples, and changes shared by AWS.

Resources

Related documents:

- [AWS Blog](#)
- [AWS Cost Management](#)
- [AWS News Blog](#)

Related examples:

- [AWS Cloud Financial Management](#)
- [AWS Well-Architected Labs: Cloud Financial Management](#)

COST01-BP09 Quantify business value from cost optimization

This best practice was updated with new guidance on December 6, 2023.

Quantifying business value from cost optimization allows you to understand the entire set of benefits to your organization. Because cost optimization is a necessary investment, quantifying business value allows you to explain the return on investment to stakeholders. Quantifying business value can help you gain more buy-in from stakeholders on future cost optimization investments, and provides a framework to measure the outcomes for your organization's cost optimization activities.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Quantifying the business value means measuring the benefits that businesses gain from the actions and decisions they take. Business value can be tangible (like reduced expenses or increased profits) or intangible (like improved brand reputation or increased customer satisfaction).

To quantify business value from cost optimization means determining how much value or benefit you're getting from your efforts to spend more efficiently. For example, if a company spends \$100,000 to deploy a workload on AWS and later optimizes it, the new cost becomes only \$80,000 without sacrificing the quality or output. In this scenario, the quantified business value from cost optimization would be a savings of \$20,000. But beyond just savings, the business might also quantify value in terms of faster delivery times, improved customer satisfaction, or other metrics that result from the cost optimization efforts. Stakeholders need to make decisions about the potential value of cost optimization, the cost of optimizing the workload, and return value.

In addition to reporting savings from cost optimization, it is recommended that you quantify the additional value delivered. Cost optimization benefits are typically quantified in terms of lower costs per business outcome. For example, you can quantify Amazon Elastic Compute Cloud(Amazon EC2) cost savings when you purchase Savings Plans, which reduce cost and maintain workload output levels. You can quantify cost reductions in AWS spending when idle Amazon EC2 instances are removed, or unattached Amazon Elastic Block Store (Amazon EBS) volumes are deleted.

The benefits from cost optimization, however, go above and beyond cost reduction or avoidance. Consider capturing additional data to measure efficiency improvements and business value.

Implementation steps

- **Evaluate business benefits:** This is the process of analyzing and adjusting AWS Cloud cost in ways that maximize the benefit received from each dollar spent. Instead of focusing on cost reduction without business value, consider business benefits and return on investments for cost optimization, which may bring more value out of the money you spend. It's about spending wisely and making investments and expenditures in areas that yield the best return.
- **Analyze forecasting AWS costs:** Forecasting helps finance stakeholders set expectations with other internal and external organization stakeholders, and can improve your organization's financial predictability. [AWS Cost Explorer](#) can be used to perform forecasting for your cost and usage.

Resources

Related documents:

- [AWS Cloud Economics](#)
- [AWS Blog](#)
- [AWS Cost Management](#)

- [AWS News Blog](#)
- [Well-Architected Reliability Pillar whitepaper](#)
- [AWS Cost Explorer](#)

Related videos:

- [Unlock Business Value with Windows on AWS](#)

Related examples:

- [Measuring and Maximizing the Business Value of Customer 360](#)
- [The Business Value of Adopting Amazon Web Services Managed Databases](#)
- [The Business Value of Amazon Web Services for Independent Software Vendors](#)
- [Business Value of Cloud Modernization](#)
- [The Business Value of Migration to Amazon Web Services](#)

Expenditure and usage awareness

Questions

- [COST 2. How do you govern usage?](#)
- [COST 3. How do you monitor your cost and usage?](#)
- [COST 4. How do you decommission resources?](#)

COST 2. How do you govern usage?

Establish policies and mechanisms to verify that appropriate costs are incurred while objectives are achieved. By employing a checks-and-balances approach, you can innovate without overspending.

Best practices

- [COST02-BP01 Develop policies based on your organization requirements](#)
- [COST02-BP02 Implement goals and targets](#)
- [COST02-BP03 Implement an account structure](#)
- [COST02-BP04 Implement groups and roles](#)

- [COST02-BP05 Implement cost controls](#)
- [COST02-BP06 Track project lifecycle](#)

COST02-BP01 Develop policies based on your organization requirements

Develop policies that define how resources are managed by your organization and inspect them periodically. Policies should cover the cost aspects of resources and workloads, including creation, modification, and decommissioning over a resource's lifetime.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Understanding your organization's costs and drivers is critical for managing your cost and usage effectively and identifying cost reduction opportunities. Organizations typically operate multiple workloads run by multiple teams. These teams can be in different organization units, each with its own revenue stream. The capability to attribute resource costs to the workloads, individual organization, or product owners drives efficient usage behaviour and helps reduce waste. Accurate cost and usage monitoring helps you understand how optimized a workload is, as well as how profitable organization units and products are. This knowledge allows for more informed decision making about where to allocate resources within your organization. Awareness of usage at all levels in the organization is key to driving change, as change in usage drives changes in cost. Consider taking a multi-faceted approach to becoming aware of your usage and expenditures.

The first step in performing governance is to use your organization's requirements to develop policies for your cloud usage. These policies define how your organization uses the cloud and how resources are managed. Policies should cover all aspects of resources and workloads that relate to cost or usage, including creation, modification, and decommissioning over a resource's lifetime. Verify that policies and procedures are followed and implemented for any change in a cloud environment. During your IT change management meetings, raise questions to find out the cost impact of planned changes, whether increasing or decreasing, the business justification, and the expected outcome.

Policies should be simple so that they are easily understood and can be implemented effectively throughout the organization. Policies also need to be easy to follow and interpret (so they are used) and specific (no misinterpretation between teams). Moreover, they need to be inspected periodically (like our mechanisms) and updated as customers business conditions or priorities change, which would make the policy outdated.

Start with broad, high-level policies, such as which geographic Region to use or times of the day that resources should be running. Gradually refine the policies for the various organizational units and workloads. Common policies include which services and features can be used (for example, lower performance storage in test and development environments), which types of resources can be used by different groups (for example, the largest size of resource in a development account is medium) and how long these resources will be in use (whether temporary, short term, or for a specific period of time).

Policy example

The following is a sample policy you can review to create your own cloud governance policies, which focus on cost optimization. Make sure you adjust policy based on your organization's requirements and your stakeholders' requests.

- **Policy name:** Define a clear policy name, such as Resource Optimization and Cost Reduction Policy.
- **Purpose:** Explain why this policy should be used and what is the expected outcome. The objective of this policy is to verify that there is a minimum cost required to deploy and run the desired workload to meet business requirements.
- **Scope:** Clearly define who should use this policy and when it should be used, such as DevOps X Team to use this policy in us-east customers for X environment (production or non-production).

Policy statement

1. Select us-east-1 or multiple us-east regions based on your workload's environment and business requirement (development, user acceptance testing, pre-production, or production).
2. Schedule Amazon EC2 and Amazon RDS instances to run between six in the morning and eight at night (Eastern Standard Time (EST)).
3. Stop all unused Amazon EC2 instances after eight hours and unused Amazon RDS instances after 24 hours of inactivity.
4. Terminate all unused Amazon EC2 instances after 24 hours of inactivity in non-production environments. Remind Amazon EC2 instance owner (based on tags) to review their stopped Amazon EC2 instances in production and inform them that their Amazon EC2 instances will be terminated within 72 hours if they are not in use.
5. Use generic instance family and size such as m5.large and then resize the instance based on CPU and memory utilization using AWS Compute Optimizer.

6. Prioritize using auto scaling to dynamically adjust the number of running instances based on traffic.
 7. Use spot instances for non-critical workloads.
 8. Review capacity requirements to commit saving plans or reserved instances for predictable workloads and inform Cloud Financial Management Team.
 9. Use Amazon S3 lifecycle policies to move infrequently accessed data to cheaper storage tiers. If no retention policy defined, use Amazon S3 Intelligent Tiering to move objects to archived tier automatically.
- 10 Monitor resource utilization and set alarms to trigger scaling events using Amazon CloudWatch.
- 11 For each AWS account, use AWS Budgets to set cost and usage budgets for your account based on cost center and business units.
- 12 Using AWS Budgets to set cost and usage budgets for your account can help you stay on top of your spending and avoid unexpected bills, allowing you to better control your costs.

Procedure: Provide detailed procedures for implementing this policy or refer to other documents that describe how to implement each policy statement. This section should provide step-by-step instructions for carrying out the policy requirements.

To implement this policy, you can use various third-party tools or AWS Config rules to check for compliance with the policy statement and trigger automated remediation actions using AWS Lambda functions. You can also use AWS Organizations to enforce the policy. Additionally, you should regularly review your resource usage and adjust the policy as necessary to verify that it continues to meet your business needs.

Implementation steps

- **Meet with stakeholders:** To develop policies, ask stakeholders (cloud business office, engineers, or functional decision makers for policy enforcement) within your organization to specify their requirements and document them. Take an iterative approach by starting broadly and continually refine down to the smallest units at each step. Team members include those with direct interest in the workload, such as organization units or application owners, as well as supporting groups, such as security and finance teams.
- **Get confirmation:** Make sure teams agree on policies who can access and deploy to the AWS Cloud. Make sure they follow your organization's policies and confirm that their resource creations align with the agreed policies and procedures.

- **Create onboarding training sessions:** Ask new organization members to complete onboarding training courses to create cost awareness and organization requirements. They may assume different policies from their previous experience or not think of them at all.
- **Define locations for your workload:** Define where your workload operates, including the country and the area within the country. This information is used for mapping to AWS Regions and Availability Zones.
- **Define and group services and resources:** Define the services that the workloads require. For each service, specify the types, the size, and the number of resources required. Define groups for the resources by function, such as application servers or database storage. Resources can belong to multiple groups.
- **Define and group the users by function:** Define the users that interact with the workload, focusing on what they do and how they use the workload, not on who they are or their position in the organization. Group similar users or functions together. You can use the AWS managed policies as a guide.
- **Define the actions:** Using the locations, resources, and users identified previously, define the actions that are required by each to achieve the workload outcomes over its life time (development, operation, and decommission). Identify the actions based on the groups, not the individual elements in the groups, in each location. Start broadly with read or write, then refine down to specific actions to each service.
- **Define the review period:** Workloads and organizational requirements can change over time. Define the workload review schedule to ensure it remains aligned with organizational priorities.
- **Document the policies:** Verify the policies that have been defined are accessible as required by your organization. These policies are used to implement, maintain, and audit access of your environments.

Resources

Related documents:

- [Change Management in the Cloud](#)
- [AWS Managed Policies for Job Functions](#)
- [AWS multiple account billing strategy](#)
- [Actions, Resources, and Condition Keys for AWS Services](#)
- [AWS Management and Governance](#)
- [Control access to AWS Regions using IAM policies](#)

- [Global Infrastructures Regions and AZs](#)

Related videos:

- [AWS Management and Governance at Scale](#)

Related examples:

- [VMware - What Are Cloud Policies?](#)

COST02-BP02 Implement goals and targets

Implement both cost and usage goals and targets for your workload. Goals provide direction to your organization on expected outcomes, and targets provide specific measurable outcomes to be achieved for your workloads.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Develop cost and usage goals and targets for your organization. As a growing organization on AWS, it is important to set and track goals for cost optimization. These goals or [key performance indicators \(KPIs\)](#) can include things like percent of spend on-demand, or adoption of certain optimized services such as AWS Graviton instances or gp3 EBS volume types. Setting measurable and achievable goals can help you to continue to measure efficiency improvements which is important to ongoing business operations. Goals provide guidance and direction to your organization on expected outcomes. Targets provide specific measurable outcomes to be achieved. In short, a goal is the direction you want to go and target is how far in that direction and when that goal should be achieved (using guidance of specific, measurable, assignable, realistic, and timely, or SMART). An example of a goal is that platform usage should increase significantly, with only a minor (non-linear) increase in cost. An example target is a 20% increase in platform usage, with less than a five percent increase in costs. Another common goal is that workloads need to be more efficient every six months. The accompanying target would be that the cost per business metrics needs to decrease by five percent every six months.

A goal for cost optimization is to increase workload efficiency, which means decreasing the cost per business outcome of the workload over time. It is recommended to implement this goal for all workloads, and also set a target such as a five percent increase in efficiency every six months

to a year. This can be achieved in the cloud through building capability in cost optimization, and releasing new services and features.

It's important to have near real-time visibility over your KPIs and related savings opportunities and track your progress over time. To get started with defining and tracking KPI goals, we recommend the KPI dashboard from the [Cloud Intelligence Dashboards \(CID\) framework](#). Based on the data from AWS Cost and Usage Report, the KPI dashboard provides a series of recommended cost optimization KPIs with the ability to set custom goals and track progress over time.

If you have another solution that allows you to set and track KPI goals, make sure it's adopted by all cloud financial management stakeholders in your organization.

Implementation steps

- **Define expected usage levels:** To begin, focus on usage levels. Engage with the application owners, marketing, and greater business teams to understand what the expected usage levels will be for the workload. How will customer demand change over time, and will there be any changes due to seasonal increases or marketing campaigns?
- **Define workload resourcing and costs:** With usage levels defined, quantify the changes in workload resources required to meet these usage levels. You may need to increase the size or number of resources for a workload component, increase data transfer, or change workload components to a different service at a specific level. Specify what the costs will be at each of these major points, and what the changes in cost will be when there are changes in usage.
- **Define business goals:** Taking the output from the expected changes in usage and cost, combine this with expected changes in technology, or any programs that you are running, and develop goals for the workload. Goals must address usage, cost and the relationship between the two. Goals must be simple, high level, and help people understand what the business expects in terms of outcomes (such as making sure unused resources are kept below certain cost level). You don't need to define goals for each unused resource type or define costs that cause losses for goals and targets. Verify that there are organizational programs (for example, capability building like training and education) if there are expected changes in cost without changes in usage.
- **Define targets:** For each of the defined goals specify a measurable target. If the goal is to increase efficiency in the workload, the target will quantify the amount of improvement (typically in business outputs for each dollar spent) and when it will be delivered. For example, if you set a goal of minimizing waste due to over-provisioning, then your target can be waste due to compute over-provisioning in the first tier of production workloads should not exceed 10% of tier compute cost, and waste due to compute over-provisioning in the second tier of production workloads should not exceed 5% of tier compute cost.

Resources

Related documents:

- [AWS managed policies for job functions](#)
- [AWS multi-account strategy for your AWS Control Tower landing zone](#)
- [Control access to AWS Regions using IAM policies](#)
- [SMART Goals](#)

Related videos:

- [Well-Architected Labs: Goals and Targets \(Level 100\)](#)

Related examples:

- [Well-Architected Labs: Decommission resources \(Goals and Targets\)](#)
- [Well-Architected Labs: Resource Type, Size, and Number \(Goals and Targets\)](#)

COST02-BP03 Implement an account structure

Implement a structure of accounts that maps to your organization. This assists in allocating and managing costs throughout your organization.

Level of risk exposed if this best practice is not established: High

Implementation guidance

AWS Organizations allows you to create multiple AWS accounts which can help you centrally govern your environment as you scale your workloads on AWS. You can model your organizational hierarchy by grouping AWS accounts in organizational unit (OU) structure and creating multiple AWS accounts under each OU. To create an account structure, you need to decide which of your AWS accounts will be the management account first. After that, you can create new AWS accounts or select existing accounts as member accounts based on your designed account structure by following [management account best practices](#) and [member account best practices](#).

It is advised to always have at least one management account with one member account linked to it, regardless of your organization size or usage. All workload resources should reside only within member accounts and no resource should be created in management account. There is

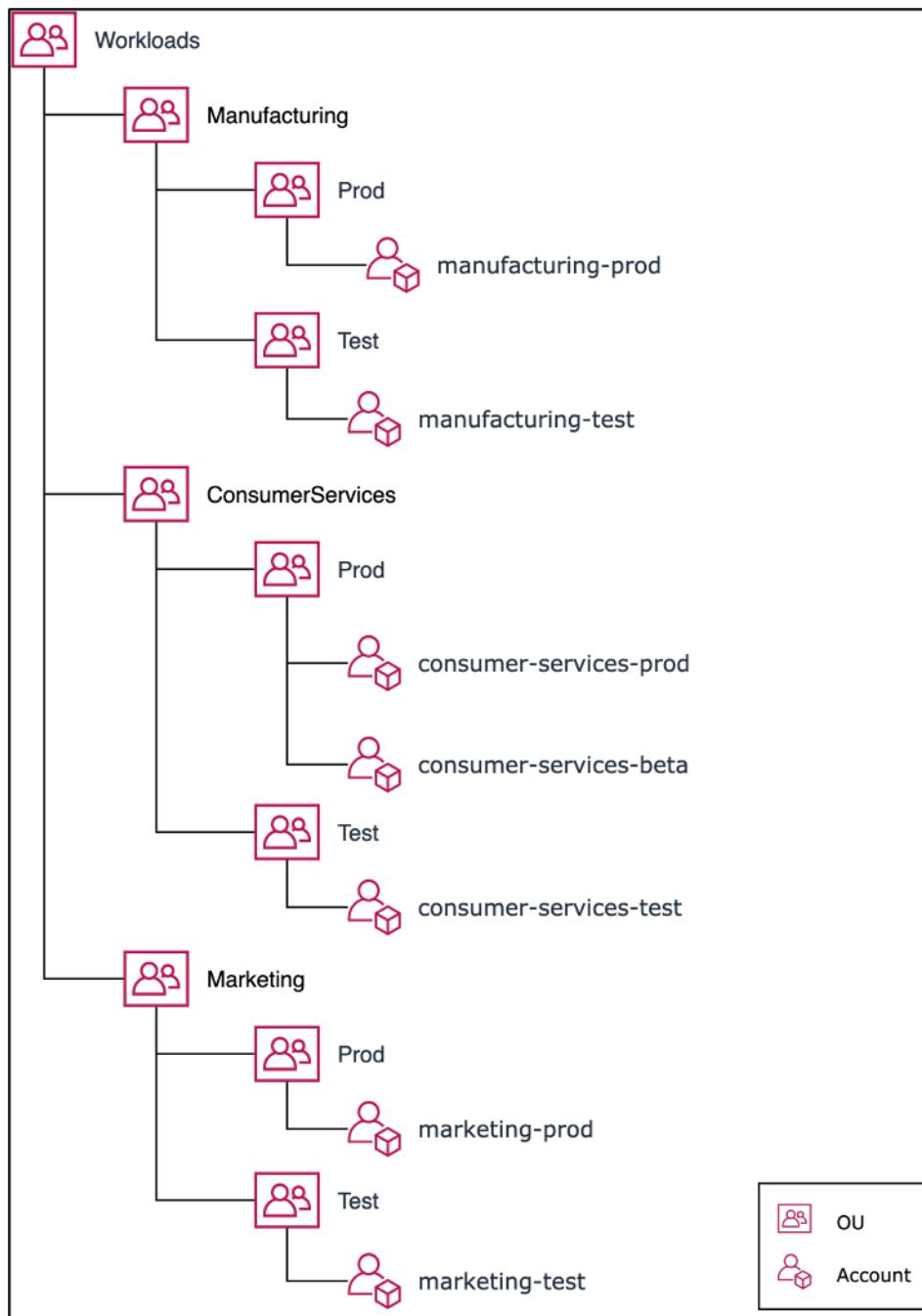
no one size fits all answer for how many AWS accounts you should have. Assess your current and future operational and cost models to ensure that the structure of your AWS accounts reflects your organization's goals. Some companies create multiple AWS accounts for business reasons, for example:

- Administrative or fiscal and billing isolation is required between organization units, cost centers, or specific workloads.
- AWS service limits are set to be specific to particular workloads.
- There is a requirement for isolation and separation between workloads and resources.

Within [AWS Organizations](#), [consolidated billing](#) creates the construct between one or more member accounts and the management account. Member accounts allow you to isolate and distinguish your cost and usage by groups. A common practice is to have separate member accounts for each organization unit (such as finance, marketing, and sales), or for each environment lifecycle (such as development, testing and production), or for each workload (workload a, b, and c), and then aggregate these linked accounts using consolidated billing.

Consolidated billing allows you to consolidate payment for multiple member AWS accounts under a single management account, while still providing visibility for each linked account's activity. As costs and usage are aggregated in the management account, this allows you to maximize your service volume discounts, and maximize the use of your commitment discounts (Savings Plans and Reserved Instances) to achieve the highest discounts.

The following diagram shows how you can use AWS Organizations with organizational units (OU) to group multiple accounts, and place multiple AWS accounts under each OU. It is recommended to use OUs for various use cases and workloads which provides patterns for organizing accounts.



Example of grouping multiple AWS accounts under organizational units.

[AWS Control Tower](#) can quickly set up and configure multiple AWS accounts, ensuring that governance is aligned with your organization's requirements.

Implementation steps

- **Define separation requirements:** Requirements for separation are a combination of multiple factors, including security, reliability, and financial constructs. Work through each factor in order

and specify whether the workload or workload environment should be separate from other workloads. Security promotes adhesion to access and data requirements. Reliability manages limits so that environments and workloads do not impact others. Review the security and reliability pillars of the Well-Architected Framework periodically and follow the provided best practices. Financial constructs create strict financial separation (different cost center, workload ownerships and accountability). Common examples of separation are production and test workloads being run in separate accounts, or using a separate account so that the invoice and billing data can be provided to the individual business units or departments in the organization or stakeholder who owns the account.

- **Define grouping requirements:** Requirements for grouping do not override the separation requirements, but are used to assist management. Group together similar environments or workloads that do not require separation. An example of this is grouping multiple test or development environments from one or more workloads together.
- **Define account structure:** Using these separations and groupings, specify an account for each group and maintain separation requirements. These accounts are your member or linked accounts. By grouping these member accounts under a single management or payer account, you combine usage, which allows for greater volume discounts across all accounts, which provides a single bill for all accounts. It's possible to separate billing data and provide each member account with an individual view of their billing data. If a member account must not have its usage or billing data visible to any other account, or if a separate bill from AWS is required, define multiple management or payer accounts. In this case, each member account has its own management or payer account. Resources should always be placed in member or linked accounts. The management or payer accounts should only be used for management.

Resources

Related documents:

- [Using Cost Allocation Tags](#)
- [AWS managed policies for job functions](#)
- [AWS multiple account billing strategy](#)
- [Control access to AWS Regions using IAM policies](#)
- [AWS Control Tower](#)
- [AWS Organizations](#)
- Best practices for [management accounts](#) and [member accounts](#)

- [Organizing Your AWS Environment Using Multiple Accounts](#)
- [Turning on shared reserved instances and Savings Plans discounts](#)
- [Consolidated billing](#)
- [Consolidated billing](#)

Related examples:

- [Splitting the CUR and Sharing Access](#)

Related videos:

- [Introducing AWS Organizations](#)
- [Set Up a Multi-Account AWS Environment that Uses Best Practices for AWS Organizations](#)

Related examples:

- [Well-Architected Labs: Create an AWS Organization \(Level 100\)](#)
- [Splitting the AWS Cost and Usage Report and Sharing Access](#)
- [Defining an AWS Multi-Account Strategy for telecommunications companies](#)
- [Best Practices for Optimizing AWS accounts](#)
- [Best Practices for Organizational Units with AWS Organizations](#)

COST02-BP04 Implement groups and roles

This best practice was updated with new guidance on December 6, 2023.

Implement groups and roles that align to your policies and control who can create, modify, or decommission instances and resources in each group. For example, implement development, test, and production groups. This applies to AWS services and third-party solutions.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

User roles and groups are fundamental building blocks in the design and implementation of secure and efficient systems. Roles and groups help organizations balance the need for control with the requirement for flexibility and productivity, ultimately supporting organizational objectives and user needs. As recommended in [Identity and access management](#) section of AWS Well-Architected Framework Security Pillar, you need robust identity management and permissions in place to provide access to the right resources for the right people under the right conditions. Users receive only the access necessary to complete their tasks. This minimizes the risk associated with unauthorized access or misuse.

After you develop policies, you can create logical groups and user roles within your organization. This allows you to assign permissions, control usage, and help implement robust access control mechanisms, preventing unauthorized access to sensitive information. Begin with high-level groupings of people. Typically, this aligns with organizational units and job roles (for example, a systems administrator in the IT Department, financial controller, or business analysts). The groups categorize people that do similar tasks and need similar access. Roles define what a group must do. It is easier to manage permissions for groups and roles than for individual users. Roles and groups assign permissions consistently and systematically across all users, preventing errors and inconsistencies.

When a user's role changes, administrators can adjust access at the role or group level, rather than reconfiguring individual user accounts. For example, a systems administrator in IT requires access to create all resources, but an analytics team member only needs to create analytics resources.

Implementation steps

- **Implement groups:** Using the groups of users defined in your organizational policies, implement the corresponding groups, if necessary. For best practices on users, groups and authentication, see the [Security Pillar](#) of the AWS Well-Architected Framework.
- **Implement roles and policies:** Using the actions defined in your organizational policies, create the required roles and access policies. For best practices on roles and policies, see the [Security Pillar](#) of the AWS Well-Architected Framework.

Resources

Related documents:

- [AWS managed policies for job functions](#)

- [AWS multiple account billing strategy](#)
- [AWS Well-Architected Framework Security Pillar](#)
- [AWS Identity and Access Management \(IAM\)](#)
- [AWS Identity and Access Management policies](#)

Related videos:

- [Why use Identity and Access Management](#)

Related examples:

- [Well-Architected Lab Basic Identity and Access](#)
- [Control access to AWS Regions using IAM policies](#)
- [Starting your Cloud Financial Management journey: Cloud cost operations](#)

COST02-BP05 Implement cost controls

Implement controls based on organization policies and defined groups and roles. These certify that costs are only incurred as defined by organization requirements such as control access to regions or resource types.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

A common first step in implementing cost controls is to set up notifications when cost or usage events occur outside of policies. You can act quickly and verify if corrective action is required without restricting or negatively impacting workloads or new activity. After you know the workload and environment limits, you can enforce governance. [AWS Budgets](#) allows you to set notifications and define monthly budgets for your AWS costs, usage, and commitment discounts (Savings Plans and Reserved Instances). You can create budgets at an aggregate cost level (for example, all costs), or at a more granular level where you include only specific dimensions such as linked accounts, services, tags, or Availability Zones.

Once you set up your budget limits with AWS Budgets, use [AWS Cost Anomaly Detection](#) to reduce your unexpected cost. AWS Cost Anomaly Detection is a cost management service that uses machine learning to continually monitor your cost and usage to detect unusual spends. It helps

you identify anomalous spend and root causes, so you can quickly take action. First, create a cost monitor in AWS Cost Anomaly Detection, then choose your alerting preference by setting up a dollar threshold (such as an alert on anomalies with impact greater than \$1,000). Once you receive alerts, you can analyze the root cause behind the anomaly and impact on your costs. You can also monitor and perform your own anomaly analysis in AWS Cost Explorer.

Enforce governance policies in AWS through [AWS Identity and Access Management](#) and [AWS Organizations Service Control Policies \(SCP\)](#). IAM allows you to securely manage access to AWS services and resources. Using IAM, you can control who can create or manage AWS resources, the type of resources that can be created, and where they can be created. This minimizes the possibility of resources being created outside of the defined policy. Use the roles and groups created previously and assign [IAM policies](#) to enforce the correct usage. SCP offers central control over the maximum available permissions for all accounts in your organization, keeping your accounts stay within your access control guidelines. SCPs are available only in an organization that has all features turned on, and you can configure the SCPs to either deny or allow actions for member accounts by default. For more details on implementing access management, see the [Well-Architected Security Pillar whitepaper](#).

Governance can also be implemented through management of [AWS service quotas](#). By ensuring service quotas are set with minimum overhead and accurately maintained, you can minimize resource creation outside of your organization's requirements. To achieve this, you must understand how quickly your requirements can change, understand projects in progress (both creation and decommission of resources), and factor in how fast quota changes can be implemented. [Service quotas](#) can be used to increase your quotas when required.

Implementation steps

- **Implement notifications on spend:** Using your defined organization policies, create [AWS Budgets](#) to notify you when spending is outside of your policies. Configure multiple cost budgets, one for each account, which notify you about overall account spending. Configure additional cost budgets within each account for smaller units within the account. These units vary depending on your account structure. Some common examples are AWS Regions, workloads (using tags), or AWS services. Configure an email distribution list as the recipient for notifications, and not an individual's email account. You can configure an actual budget for when an amount is exceeded, or use a forecasted budget for notifying on forecasted usage. You can also preconfigure AWS Budget Actions that can enforce specific IAM or SCP policies, or stop target Amazon EC2 or Amazon RDS instances. Budget Actions can be started automatically or require workflow approval.

- **Implement notifications on anomalous spend:** Use [AWS Cost Anomaly Detection](#) to reduce your surprise costs in your organization and analyze root cause of potential anomalous spend. Once you create cost monitor to identify unusual spend at your specified granularity and configure notifications in AWS Cost Anomaly Detection, it sends you alert when unusual spend is detected. This will allow you to analyze root case behind the anomaly and understand the impact on your cost. Use AWS Cost Categories while configuring AWS Cost Anomaly Detection to identify which project team or business unit team can analyze the root cause of the unexpected cost and take timely necessary actions.
- **Implement controls on usage:** Using your defined organization policies, implement IAM policies and roles to specify which actions users can perform and which actions they cannot. Multiple organizational policies may be included in an AWS policy. In the same way that you defined policies, start broadly and then apply more granular controls at each step. Service limits are also an effective control on usage. Implement the correct service limits on all your accounts.

Resources

Related documents:

- [AWS managed policies for job functions](#)
- [AWS multiple account billing strategy](#)
- [Control access to AWS Regions using IAM policies](#)
- [AWS Budgets](#)
- [AWS Cost Anomaly Detection](#)
- [Control Your AWS Costs](#)

Related videos:

- [How can I use AWS Budgets to track my spending and usage](#)

Related examples:

- [Example IAM access management policies](#)
- [Example service control policies](#)
- [AWS Budgets Actions](#)
- [Create IAM Policy to control access to Amazon EC2 resources using Tags](#)

- [Restrict the access of IAM Identity to specific Amazon EC2 resources](#)
- [Create an IAM Policy to restrict Amazon EC2 usage by family](#)
- [Well-Architected Labs: Cost and Usage Governance \(Level 100\)](#)
- [Well-Architected Labs: Cost and Usage Governance \(Level 200\)](#)
- [Slack integrations for Cost Anomaly Detection using AWS Chatbot](#)

COST02-BP06 Track project lifecycle

This best practice was updated with new guidance on December 6, 2023.

Track, measure, and audit the lifecycle of projects, teams, and environments to avoid using and paying for unnecessary resources.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Tracking project lifecycle effectively allows organizations to have better control over costs through improved planning, management, and optimization of resources, time, and quality. The insights gained through tracking are invaluable for making informed decisions that contribute to the cost-effectiveness and overall success of the project.

Tracking the entire lifecycle of the workload helps you understand when workloads or workload components are no longer required. The existing workloads and components may appear to be in use, but when AWS releases new services or features, they can be decommissioned or adopted. Check the previous stages of workloads. After a workload is in production, previous environments can be decommissioned or greatly reduced in capacity until they are required again.

AWS provides a number of management and governance services you can use for entity lifecycle tracking. You can use [AWS Config](#) or [AWS Systems Manager](#) to provide a detailed inventory of your AWS resources and configuration. It is recommended that you integrate with your existing project or asset management systems to keep track of active projects and products within your organization. Combining your current system with the rich set of events and metrics provided by AWS allows you to build a view of significant lifecycle events and proactively manage resources to reduce unnecessary costs.

Similar to [Application Lifecycle Management \(ALM\)](#), tracking project lifecycle should involve multiple processes, tools, and teams working together, such as design and development, testing, production, support, and workload redundancy.

By carefully monitoring each phase of a project's lifecycle, organizations gain crucial insights and enhanced control, facilitating successful project planning, implementation, and completion. This careful oversight verifies that projects not only meet quality standards, but are delivered on time and within budget, promoting overall cost efficiency.

For more details on implementing entity lifecycle tracking, see the [AWS Well-Architected Operational Excellence Pillar whitepaper](#).

Implementation steps

- **Establish a project lifecycle monitoring process:** The [Cloud Center of Excellence team](#) must establish project lifecycle monitoring process. Establish a structured and systematic approach to monitoring workloads in order to improve control, visibility, and performance of the projects. Make the monitoring process transparent, collaborative, and focused on continuous improvement to maximize its effectiveness and value.
- **Perform workload reviews:** As defined by your organizational policies, set up a regular cadence to audit your existing projects and perform workload reviews. The amount of effort spent in the audit should be proportional to the approximate risk, value, or cost to the organization. Key areas to include in the audit would be risk to the organization of an incident or outage, value, or contribution to the organization (measured in revenue or brand reputation), cost of the workload (measured as total cost of resources and operational costs), and usage of the workload (measured in number of organization outcomes per unit of time). If these areas change over the lifecycle, adjustments to the workload are required, such as full or partial decommissioning.

Resources

Related documents:

- [Guidance for Tagging on AWS](#)
- [What Is ALM \(Application Lifecycle Management\)?](#)
- [AWS managed policies for job functions](#)

Related examples:

- [Control access to AWS Regions using IAM policies](#)

Related tools:

- [AWS Config](#)
- [AWS Systems Manager](#)
- [AWS Budgets](#)
- [AWS Organizations](#)
- [AWS CloudFormation](#)

COST 3. How do you monitor your cost and usage?

Establish policies and procedures to monitor and appropriately allocate your costs. This permits you to measure and improve the cost efficiency of this workload.

Best practices

- [COST03-BP01 Configure detailed information sources](#)
- [COST03-BP02 Add organization information to cost and usage](#)
- [COST03-BP03 Identify cost attribution categories](#)
- [COST03-BP04 Establish organization metrics](#)
- [COST03-BP05 Configure billing and cost management tools](#)
- [COST03-BP06 Allocate costs based on workload metrics](#)

COST03-BP01 Configure detailed information sources

Configure the cost management and reporting tools for hourly granularity to provide detailed cost and usage information, enabling deeper analytics and transparency. Configure your workload to generate or have the log entries for every delivered business outcome.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Detailed billing information such as hourly granularity in cost management tools allow organizations to track their consumptions with further details and help them to identify some of

the cost increase reasons. These data sources provide the most accurate view of cost and usage across your entire organization.

AWS Cost and Usage Report provides daily or hourly usage granularity, rates, costs, and usage attributes for all chargeable AWS services. All possible dimensions are in the CUR, including tagging, location, resource attributes, and account IDs.

Configure your CUR with the following customizations:

- Include resource IDs
- Automatically refresh the CUR
- Hourly granularity
- **Versioning:** Overwrite existing report
- **Data integration:** Athena (Parquet format and compression)

Use [AWS Glue](#) to prepare the data for analysis, and use [Amazon Athena](#) to perform data analysis, using SQL to query the data. You can also use [Amazon QuickSight](#) to build custom and complex visualizations and distribute them throughout your organization.

Implementation steps

- **Configure the cost and usage report:** Using the billing console, configure at least one cost and usage report. Configure a report with hourly granularity that includes all identifiers and resource IDs. You can also create other reports with different granularities to provide higher-level summary information.
- **Configure hourly granularity in Cost Explorer:** Enable **Hourly and Resource Level Data** to access cost and usage data at hourly granularity for the past 14 days and resource level granularity.
- **Configure application logging:** Verify that your application logs each business outcome that it delivers so it can be tracked and measured. Ensure that the granularity of this data is at least hourly so it matches with the cost and usage data. For more details on logging and monitoring, see [Well-Architected Operational Excellence Pillar](#).

Resources

Related documents:

- [AWS Cost and Usage Report](#)
- [AWS Glue](#)
- [Amazon QuickSight](#)
- [AWS Cost Management Pricing](#)
- [Tagging AWS resources](#)
- [Analyzing your costs with AWS Budgets](#)
- [Analyzing your costs with Cost Explorer](#)
- [Managing AWS Cost and Usage Reports](#)
- [Well-Architected Operational Excellence Pillar](#)

Related examples:

- [AWS Account Setup](#)
- [AWS Cost Explorer's New Look and Common Use Cases](#)

COST03-BP02 Add organization information to cost and usage

Define a tagging schema based on your organization, workload attributes, and cost allocation categories so that you can filter and search for resources or monitor cost and usage in cost management tools. Implement consistent tagging across all resources where possible by purpose, team, environment, or other criteria relevant to your business.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Implement [tagging in AWS](#) to add organization information to your resources, which will then be added to your cost and usage information. A tag is a key-value pair — the key is defined and must be unique across your organization, and the value is unique to a group of resources. An example of a key-value pair is the key is Environment, with a value of Production. All resources in the production environment will have this key-value pair. Tagging allows you categorize and track your costs with meaningful, relevant organization information. You can apply tags that represent organization categories (such as cost centers, application names, projects, or owners), and identify workloads and characteristics of workloads (such as test or production) to attribute your costs and usage throughout your organization.

When you apply tags to your AWS resources (such as Amazon Elastic Compute Cloud instances or Amazon Simple Storage Service buckets) and activate the tags, AWS adds this information to your Cost and Usage Reports. You can run reports and perform analysis on tagged and untagged resources to allow greater compliance with internal cost management policies and ensure accurate attribution.

Creating and implementing an AWS tagging standard across your organization's accounts helps you manage and govern your AWS environments in a consistent and uniform manner. Use [Tag Policies](#) in AWS Organizations to define rules for how tags can be used on AWS resources in your accounts in AWS Organizations. Tag Policies allow you to easily adopt a standardized approach for tagging AWS resources

[AWS Tag Editor](#) allows you to add, delete, and manage tags of multiple resources. With Tag Editor, you search for the resources that you want to tag, and then manage tags for the resources in your search results.

[AWS Cost Categories](#) allows you to assign organization meaning to your costs, without requiring tags on resources. You can map your cost and usage information to unique internal organization structures. You define category rules to map and categorize costs using billing dimensions, such as accounts and tags. This provides another level of management capability in addition to tagging. You can also map specific accounts and tags to multiple projects.

Implementation steps

- **Define a tagging schema:** Gather all stakeholders from across your business to define a schema. This typically includes people in technical, financial, and management roles. Define a list of tags that all resources must have, as well as a list of tags that resources should have. Verify that the tag names and values are consistent across your organization.
- **Tag resources:** Using your defined cost attribution categories, [place tags](#) on all resources in your workloads according to the categories. Use tools such as the CLI, Tag Editor, or AWS Systems Manager to increase efficiency.
- **Implement AWS Cost Categories:** You can create [Cost Categories](#) without implementing tagging. Cost categories use the existing cost and usage dimensions. Create category rules from your schema and implement them into cost categories.
- **Automate tagging:** To verify that you maintain high levels of tagging across all resources, automate tagging so that resources are automatically tagged when they are created. Use services such as [AWS CloudFormation](#) to verify that resources are tagged when created. You can also create a custom solution to [tag automatically](#) using Lambda functions or use a microservice that

scans the workload periodically and removes any resources that are not tagged, which is ideal for test and development environments.

- **Monitor and report on tagging:** To verify that you maintain high levels of tagging across your organization, report and monitor the tags across your workloads. You can use [AWS Cost Explorer](#) to view the cost of tagged and untagged resources, or use services such as [Tag Editor](#). Regularly review the number of untagged resources and take action to add tags until you reach the desired level of tagging.

Resources

Related documents:

- [Tagging Best Practices](#)
- [AWS CloudFormation Resource Tag](#)
- [AWS Cost Categories](#)
- [Tagging AWS resources](#)
- [Analyzing your costs with AWS Budgets](#)
- [Analyzing your costs with Cost Explorer](#)
- [Managing AWS Cost and Usage Reports](#)

Related videos:

- [How can I tag my AWS resources to divide up my bill by cost center or project](#)
- [Tagging AWS Resources](#)

Related examples:

- [Automatically tag new AWS resources based on identity or role](#)

COST03-BP03 Identify cost attribution categories

Identify organization categories such as business units, departments or projects that could be used to allocate cost within your organization to the internal consuming entities. Use those categories to enforce spend accountability, create cost awareness and drive effective consumption behaviors.

Level of risk exposed if this best practice is not established: High

Implementation guidance

The process of categorizing costs is crucial in budgeting, accounting, financial reporting, decision making, benchmarking, and project management. By classifying and categorizing expenses, teams can gain a better understanding of the types of costs they incur throughout their cloud journey helping teams make informed decisions and manage budgets effectively.

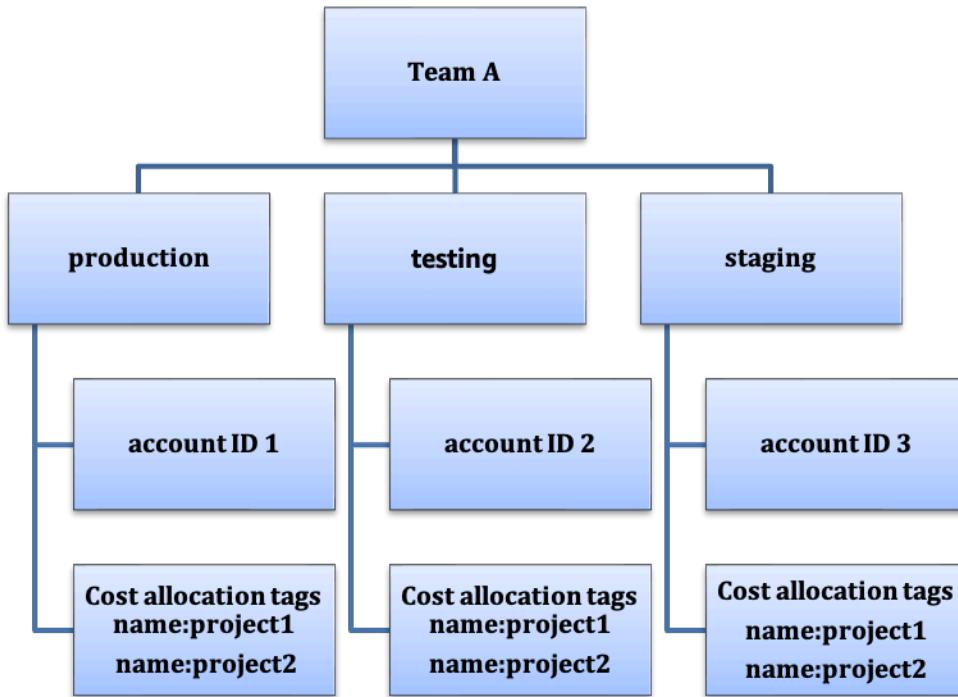
Cloud spend accountability establishes a strong incentive for disciplined demand and cost management. The result is significantly greater cloud cost savings for organizations that allocate most of their cloud spend to consuming business units or teams. Moreover, allocating cloud spend helps organizations adopt more best practices of centralized cloud governance.

Work with your finance team and other relevant stakeholders to understand the requirements of how costs must be allocated within your organization during your regular cadence calls. Workload costs must be allocated throughout the entire lifecycle, including development, testing, production, and decommissioning. Understand how the costs incurred for learning, staff development, and idea creation are attributed in the organization. This can be helpful to correctly allocate accounts used for this purpose to training and development budgets instead of generic IT cost budgets.

After defining your cost attribution categories with stakeholders in your organization, use [AWS Cost Categories](#) to group your cost and usage information into meaningful categories in the AWS Cloud, such as cost for a specific project, or AWS accounts for departments or business units. You can create custom categories and map your cost and usage information into these categories based on rules you define using various dimensions such as account, tag, service, or charge type. Once cost categories are set up, you can view your cost and usage information by these categories, which allows your organization to make better strategic and purchasing decisions. These categories are visible in AWS Cost Explorer, AWS Budgets, and AWS Cost and Usage Report as well.

For example, create cost categories for your business units (DevOps team), and under each category create multiple rules (rules for each sub category) with multiple dimensions (AWS accounts, cost allocation tags, services or charge type) based on your defined groupings. With cost categories, you can organize your costs using a rule-based engine. The rules that you configure organize your costs into categories. Within these rules, you can filter with using multiple dimensions for each category such as specific AWS accounts, AWS services, or charge types. You can then use these categories across multiple products in the [AWS Billing and Cost Management and Cost Management console](#). This includes AWS Cost Explorer, AWS Budgets, AWS Cost and Usage Report, and AWS Cost Anomaly Detection.

As an example, the following diagram displays how to group your costs and usage information in your organization by having multiple teams (cost category), multiple environments (rules), and each environment having multiple resources or assets (dimensions).



Cost and usage organization chart

You can create groupings of costs using cost categories as well. After you create the cost categories (allowing up to 24 hours after creating a cost category for your usage records to be updated with values), they appear in [AWS Cost Explorer](#), [AWS Budgets](#), [AWS Cost and Usage Report](#), and [AWS Cost Anomaly Detection](#). In AWS Cost Explorer and AWS Budgets, a cost category appears as an additional billing dimension. You can use this to filter for the specific cost category value, or group by the cost category.

Implementation steps

- **Define your organization categories:** Meet with internal stakeholders and business units to define categories that reflect your organization's structure and requirements. These categories should directly map to the structure of existing financial categories, such as business unit, budget, cost center, or department. Look at the outcomes the cloud delivers for your business such as training or education, as these are also organization categories.
- **Define your functional categories:** Meet with internal stakeholders and business units to define categories that reflect the functions that you have within your business. This may be the

workload or application names, and the type of environment, such as production, testing, or development.

- **Define AWS Cost Categories:** Create cost categories to organize your cost and usage information with using [AWS Cost Categories](#) and map your AWS cost and usage into [meaningful categories](#). Multiple categories can be assigned to a resource, and a resource can be in multiple different categories, so define as many categories as needed so that you can [manage your costs](#) within the categorized structure using AWS Cost Categories.

Resources

Related documents:

- [Tagging AWS resources](#)
- [Using Cost Allocation Tags](#)
- [Analyzing your costs with AWS Budgets](#)
- [Analyzing your costs with Cost Explorer](#)
- [Managing AWS Cost and Usage Reports](#)
- [AWS Cost Categories](#)
- [Managing your costs with AWS Cost Categories](#)
- [Creating cost categories](#)
- [Tagging cost categories](#)
- [Splitting charges within cost categories](#)
- [AWS Cost Categories Features](#)

Related examples:

- [Organize your cost and usage data with AWS Cost Categories](#)
- [Managing your costs with AWS Cost Categories](#)
- [Well-Architected Labs: Cost and Usage Visualization](#)
- [Well-Architected Labs: Cost Categories](#)

COST03-BP04 Establish organization metrics

Establish the organization metrics that are required for this workload. Example metrics of a workload are customer reports produced, or web pages served to customers.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Understand how your workload's output is measured against business success. Each workload typically has a small set of major outputs that indicate performance. If you have a complex workload with many components, then you can prioritize the list, or define and track metrics for each component. Work with your teams to understand which metrics to use. This unit will be used to understand the efficiency of the workload, or the cost for each business output.

Implementation steps

- **Define workload outcomes:** Meet with the stakeholders in the business and define the outcomes for the workload. These are a primary measure of customer usage and must be business metrics and not technical metrics. There should be a small number of high-level metrics (less than five) per workload. If the workload produces multiple outcomes for different use cases, then group them into a single metric.
- **Define workload component outcomes:** Optionally, if you have a large and complex workload, or can easily break your workload into components (such as microservices) with well-defined inputs and outputs, define metrics for each component. The effort should reflect the value and cost of the component. Start with the largest components and work towards the smaller components.

Resources

Related documents:

- [Tagging AWS resources](#)
- [Analyzing your costs with AWS Budgets](#)
- [Analyzing your costs with Cost Explorer](#)
- [Managing AWS Cost and Usage Reports](#)

COST03-BP05 Configure billing and cost management tools

Configure cost management tools in line with your organization policies to manage and optimize cloud spend. This includes services, tools, and resources to organize and track cost and usage data, enhance control through consolidated billing and access permission, improve planning through budgeting and forecasts, receive notifications or alerts, and further lower cost with resources and pricing optimizations.

Level of risk exposed if this best practice is not established: High

Implementation guidance

To establish strong accountability, your account strategy should be considered first as part of your cost allocation strategy. Get this right, and you may not need to go any further. Otherwise, there can be unawareness and further pain points.

To encourage accountability of cloud spend, users should have access to tools that provide visibility into their costs and usage. It is recommended that all workloads and teams have tools configured for the following details and purposes:

- **Organize:** Establish your cost allocation and governance baseline with your own tagging strategy and taxonomy. Tag supported AWS resources and categorize them meaningfully based on your organization structure (business units, departments, or projects). Tag account names for specific cost centers and map them with AWS Cost Categories to group accounts for particular business units to their cost centers so that business unit owner can see multiple accounts' consumption in one place.
- **Access:** Track organization-wide billing information in [consolidated billing](#) and verify the right stakeholders and business owners have access.
- **Control:** Build effective governance mechanisms with the right guardrails to prevent unexpected scenarios when using Service Control Policies (SCP), tag policies, and budget alerts. For example, you can allow teams to create resources in preferred Regions only by using effective control mechanisms.
- **Current State:** Configure a dashboard showing current levels of cost and usage. The dashboard should be available in a highly visible place within the work environment similar to an operations dashboard. You can use [Cloud Intelligence Dashboard \(CID\)](#) or any other supported products to create this visibility.
- **Notifications:** Provide notifications when cost or usage is outside of defined limits and when anomalies occur with AWS Budgets or AWS Cost Anomaly Detection.

- **Reports:** Summarize all cost and usage information and raise awareness and accountability of your cloud spend with detailed, attributable cost data. Reports should be relevant to the team consuming them and ideally should contain recommendations.
- **Tracking:** Show the current cost and usage against configured goals or targets.
- **Analysis:** Allow team members to perform custom and deep analysis down to the hourly granularity, with all possible dimensions.
- **Inspect:** Stay up to date with your resource deployment and cost optimization opportunities. Get notifications (using Amazon CloudWatch, Amazon SNS, or Amazon SES) for resource deployments at the organization level and review cost optimization recommendations (for example, AWS Compute Optimizer or AWS Trusted Advisor).
- **Trending:** Display the variability in cost and usage over the required period of time, with the required granularity.
- **Forecasts:** Show estimated future costs, estimate your resource usage, and spend with forecast dashboards that you create.

You can use AWS tools like [AWS Cost Explorer](#), [AWS Billing and Cost Management](#), or [AWS Budgets](#) for essentials, or you can integrate CUR data with [Amazon Athena](#) and [Amazon QuickSight](#) to provide this capability for more detailed views. If you don't have essential skills or bandwidth in your organization, you can work with [AWS ProServ](#), [AWS Managed Services \(AMS\)](#), or [AWS Partners](#) and use their tools. You can also use third-party tools, but verify first that the cost provides value to your organization.

Implementation steps

- **Allow team-based access to tools:** Configure your accounts and create groups that have access to the required cost and usage reports for their consumptions and use [AWS Identity and Access Management](#) to [control access](#) to the tools such as AWS Cost Explorer. These groups must include representatives from all teams that own or manage an application. This certifies that every team has access to their cost and usage information to track their consumption.
- **Configure AWS Budgets:** [Configure AWS Budgets](#) on all accounts for your workloads. Set budgets for the overall account spend, and budgets for the workloads by using tags. Configure notifications in AWS Budgets to receive alerts for when you exceed your budgeted amounts, or when your estimated costs exceed your budgets.
- **Configure AWS Cost Explorer:** Configure [AWS Cost Explorer](#) for your workload and accounts to visualize your cost data for further analysis. Create a dashboard for the workload that tracks

overall spend, key usage metrics for the workload, and forecast of future costs based on your historical cost data.

- **Configure AWS Cost Anomaly Detection:** Use [AWS Cost Anomaly Detection](#) for your accounts, core services or cost categories you created to monitor your cost and usage and detect unusual spends. You can receive alerts individually in aggregated reports, and receive alerts in an email or an Amazon SNS topic which allows you to analyze and determine the root cause of the anomaly, and identify the factor that is driving the cost increase.
- **Configure advanced tools:** Optionally, you can create custom tools for your organization that provides additional detail and granularity. You can implement advanced analysis capability using [Amazon Athena](#), and dashboards using [Amazon QuickSight](#). Consider using the [CID solution](#) which has pre-configured, advanced dashboards. There are also [AWS Partners](#) you can work with and adopt their cloud management solutions to enable cloud bill monitoring and optimization in one convenient location.

Resources

Related documents:

- [AWS Cost Management](#)
- [Tagging AWS resources](#)
- [Analyzing your costs with AWS Budgets](#)
- [Analyzing your costs with Cost Explorer](#)
- [Managing AWS Cost and Usage Report](#)
- [AWS Cost Categories](#)
- [Cloud Financial Management with AWS](#)
- [Example service control policies](#)
- [AWS APN Partners – Cost Management](#)

Related videos:

- [Deploying Cloud Intelligence Dashboards](#)
- [Get Alerts on any FinOps or Cost Optimization Metric or KPI](#)

Related examples:

- [Well-Architected Labs - AWS Account Setup](#)
- [Well-Architected Labs: Billing Visualization](#)
- [Well-Architected Labs: Cost and Governance Usage](#)
- [Well-Architected Labs: Cost and Usage Analysis](#)
- [Well-Architected Labs: Cost and Usage Visualization](#)
- [Well-Architected Labs: Cloud Intelligence Dashboards](#)
- [How to use SCPs to set permission guardrails across accounts](#)

COST03-BP06 Allocate costs based on workload metrics

Allocate the workload's costs by usage metrics or business outcomes to measure workload cost efficiency. Implement a process to analyze the cost and usage data with analytics services, which can provide insight and charge back capability.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Cost optimization is delivering business outcomes at the lowest price point, which can only be achieved by allocating workload costs by workload metrics (measured by workload efficiency). Monitor the defined workload metrics through log files or other application monitoring. Combine this data with the workload costs, which can be obtained by looking at costs with a specific tag value or account ID. It is recommended to perform this analysis at the hourly level. Your efficiency will typically change if you have some static cost components (for example, a backend database running permanently) with a varying request rate (for example, usage peaks at nine in the morning to five in the evening, with few requests at night). Understanding the relationship between the static and variable costs will help you to focus your optimization activities.

Creating workload metrics for shared resources may be challenging compared to resources like containerized applications on Amazon Elastic Container Service (Amazon ECS) and Amazon API Gateway. However, there are certain ways you can categorize usage and track cost. If you need to track Amazon ECS and AWS Batch shared resources, you can enable split cost allocation data in AWS Cost Explorer. With split cost allocation data, you can understand and optimize the cost and usage of your containerized applications and allocate application costs back to individual business entities based on how shared compute and memory resources are consumed. If you have shared API Gateway and AWS Lambda function usage, then you can use [AWS Application Cost Profiler](#) to categorize their consumption based on their Tenant ID or Customer ID.

Implementation steps

- **Allocate costs to workload metrics:** Using the defined metrics and configured tags, create a metric that combines the workload output and workload cost. Use analytics services such as Amazon Athena and Amazon QuickSight to create an efficiency dashboard for the overall workload and any components.

Resources

Related documents:

- [Tagging AWS resources](#)
- [Analyzing your costs with AWS Budgets](#)
- [Analyzing your costs with Cost Explorer](#)
- [Managing AWS Cost and Usage Reports](#)

Related examples:

- [Improve cost visibility of Amazon ECS and AWS Batch with AWS Split Cost Allocation Data](#)

COST 4. How do you decommission resources?

Implement change control and resource management from project inception to end-of-life. This ensures you shut down or terminates unused resources to reduce waste.

Best practices

- [COST04-BP01 Track resources over their lifetime](#)
- [COST04-BP02 Implement a decommissioning process](#)
- [COST04-BP03 Decommission resources](#)
- [COST04-BP04 Decommission resources automatically](#)
- [COST04-BP05 Enforce data retention policies](#)

COST04-BP01 Track resources over their lifetime

Define and implement a method to track resources and their associations with systems over their lifetime. You can use tagging to identify the workload or function of the resource.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Decommission workload resources that are no longer required. A common example is resources used for testing: after testing has been completed, the resources can be removed. Tracking resources with tags (and running reports on those tags) can help you identify assets for decommission, as they will not be in use or the license on them will expire. Using tags is an effective way to track resources, by labeling the resource with its function, or a known date when it can be decommissioned. Reporting can then be run on these tags. Example values for feature tagging are feature-X testing to identify the purpose of the resource in terms of the workload lifecycle. Another example is using LifeSpan or TTL for the resources, such as to-be-deleted tag key name and value to define the time period or specific time for decommissioning.

Implementation steps

- **Implement a tagging scheme:** Implement a tagging scheme that identifies the workload the resource belongs to, verifying that all resources within the workload are tagged accordingly. Tagging helps you categorize resources by purpose, team, environment, or other criteria relevant to your business. For more detail on tagging uses cases, strategies, and techniques, see [AWS Tagging Best Practices](#).
- **Implement workload throughput or output monitoring:** Implement workload throughput monitoring or alarming, initiating on either input requests or output completions. Configure it to provide notifications when workload requests or outputs drop to zero, indicating the workload resources are no longer used. Incorporate a time factor if the workload periodically drops to zero under normal conditions. For more detail on unused or underutilized resources, see [AWS Trusted Advisor Cost Optimization checks](#).
- **Group AWS resources:** Create groups for AWS resources. You can use [AWS Resource Groups](#) to organize and manage your AWS resources that are in the same AWS Region. You can add tags to most of your resources to help identify and sort your resources within your organization. Use [Tag Editor](#) add tags to supported resources in bulk. Consider using [AWS Service Catalog](#) to create, manage, and distribute portfolios of approved products to end users and manage the product lifecycle.

Resources

Related documents:

- [AWS Auto Scaling](#)
- [AWS Trusted Advisor](#)
- [AWS Trusted Advisor Cost Optimization Checks](#)
- [Tagging AWS resources](#)
- [Publishing Custom Metrics](#)

Related videos:

- [How to optimize costs using AWS Trusted Advisor](#)

Related examples:

- [Organize AWS resources](#)
- [Optimize cost using AWS Trusted Advisor](#)

COST04-BP02 Implement a decommissioning process

Implement a process to identify and decommission unused resources.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Implement a standardized process across your organization to identify and remove unused resources. The process should define the frequency searches are performed and the processes to remove the resource to verify that all organization requirements are met.

Implementation steps

- **Create and implement a decommissioning process:** Work with the workload developers and owners to build a decommissioning process for the workload and its resources. The process should cover the method to verify if the workload is in use, and also if each of the workload resources are in use. Detail the steps necessary to decommission the resource, removing them from service while ensuring compliance with any regulatory requirements. Any associated resources should be included, such as licenses or attached storage. Notify the workload owners that the decommissioning process has been started.

Use the following decommission steps to guide you on what should be checked as part of your process:

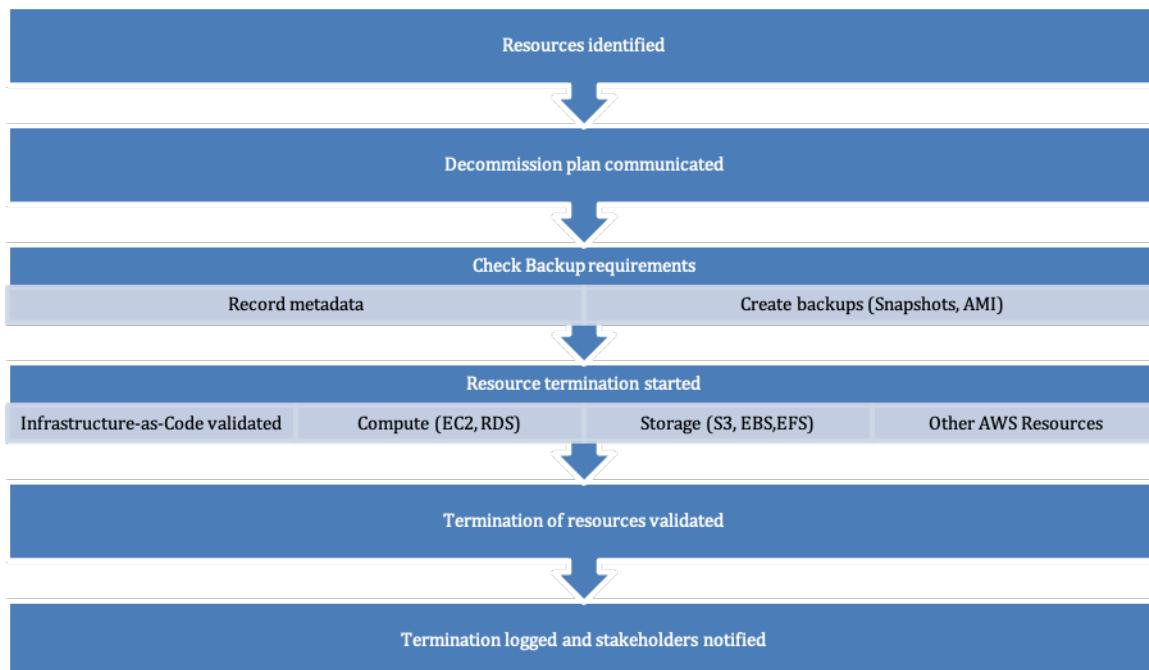
- **Identify resources to be decommissioned:** Identify resources that are eligible for decommissioning in your AWS Cloud. Record all necessary information and schedule the decommission. In your timeline, be sure to account for if (and when) unexpected issues arise during the process.
- **Coordinate and communicate:** Work with workload owners to confirm the resource to be decommissioned
- **Record metadata and create backups:** Record metadata (such as public IPs, Region, AZ, VPC, Subnet, and Security Groups) and create backups (such as Amazon Elastic Block Store snapshots or taking AMI, keys export, and Certificate export) if it is required for the resources in the production environment or if they are critical resources.
- **Validate infrastructure-as-code:** Determine whether resources were deployed with AWS CloudFormation, Terraform, AWS Cloud Development Kit (AWS CDK), or any other infrastructure-as-code deployment tool so they can be re-deployed if necessary.
- **Prevent access:** Apply restrictive controls for a period of time, to prevent the use of resources while you determine if the resource is required. Verify that the resource environment can be reverted to its original state if required.
- **Follow your internal decommissioning process:** Follow the administrative tasks and decommissioning process of your organization, like removing the resource from your organization domain, removing the DNS record, and removing the resource from your configuration management tool, monitoring tool, automation tool and security tools.

If the resource is an Amazon EC2 instance, consult the following list. [For more detail, see How do I delete or terminate my Amazon EC2 resources?](#)

- Stop or terminate all your Amazon EC2 instances and load balancers. Amazon EC2 instances are visible in the console for a short time after they're terminated. You aren't billed for any instances that aren't in the running state
- Delete your Auto Scaling infrastructure.
- Release all Dedicated Hosts.
- Delete all Amazon EBS volumes and Amazon EBS snapshots.
- Release all Elastic IP addresses.
- Deregister all Amazon Machine Images (AMIs).
- Terminate all AWS Elastic Beanstalk environments.

If the resource is an object in Amazon S3 Glacier storage and if you delete an archive before meeting the minimum storage duration, you will be charged a prorated early deletion fee. Amazon S3 Glacier minimum storage duration depends on the storage class used. For a summary of minimum storage duration for each storage class, see [Performance across the Amazon S3 storage classes](#). For detail on how early deletion fees are calculated, see [Amazon S3 pricing](#).

The following simple decommissioning process flowchart outlines the decommissioning steps. Before decommissioning resources, verify that resources you have identified for decommissioning are not being used by the organization.



Resource decommissioning flow.

Resources

Related documents:

- [AWS Auto Scaling](#)
- [AWS Trusted Advisor](#)
- [AWS CloudTrail](#)

Related videos:

- [Delete CloudFormation stack but retain some resources](#)
- [Find out which user launched Amazon EC2 instance](#)

Related examples:

- [Delete or terminate Amazon EC2 resources](#)
- [Find out which user launched an Amazon EC2 instance](#)

COST04-BP03 Decommission resources

Decommission resources initiated by events such as periodic audits, or changes in usage. Decommissioning is typically performed periodically and can be manual or automated.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

The frequency and effort to search for unused resources should reflect the potential savings, so an account with a small cost should be analyzed less frequently than an account with larger costs. Searches and decommission events can be initiated by state changes in the workload, such as a product going end of life or being replaced. Searches and decommission events may also be initiated by external events, such as changes in market conditions or product termination.

Implementation steps

- **Decommission resources:** This is the depreciation stage of AWS resources that are no longer needed or ending of a licensing agreement. Complete all final checks completed before moving to the disposal stage and decommissioning resources to prevent any unwanted disruptions like taking snapshots or backups. Using the decommissioning process, decommission each of the resources that have been identified as unused.

Resources

Related documents:

- [AWS Auto Scaling](#)
- [AWS Trusted Advisor](#)

Related examples:

- [Well-Architected Labs: Decommission resources \(Level 100\)](#)

COST04-BP04 Decommission resources automatically

Design your workload to gracefully handle resource termination as you identify and decommission non-critical resources, resources that are not required, or resources with low utilization.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Use automation to reduce or remove the associated costs of the decommissioning process.

Designing your workload to perform automated decommissioning will reduce the overall workload costs during its lifetime. You can use [Amazon EC2 Auto Scaling](#) or [Application Auto Scaling](#) to perform the decommissioning process. You can also implement custom code using the [API or SDK](#) to decommission workload resources automatically.

[Modern applications](#) are built serverless-first, a strategy that prioritizes the adoption of serverless services. AWS developed [serverless services](#) for all three layers of your stack: compute, integration, and data stores. Using serverless architecture will allow you to save costs during low-traffic periods with scaling up and down automatically.

Implementation steps

- **Implement Amazon EC2 Auto Scaling or Application Auto Scaling:** For resources that are supported, configure them with Amazon EC2 Auto Scaling or Application Auto Scaling. These services can help you optimize your utilization and cost efficiencies when consuming AWS services. When demand drops, these services will automatically remove any excess resource capacity so you avoid overspending.
- **Configure CloudWatch to terminate instances:** Instances can be configured to terminate using [CloudWatch alarms](#). Using the metrics from the decommissioning process, implement an alarm with an Amazon Elastic Compute Cloud action. Verify the operation in a non-production environment before rolling out.
- **Implement code within the workload:** You can use the AWS SDK or AWS CLI to decommission workload resources. Implement code within the application that integrates with AWS and terminates or removes resources that are no longer used.

- **Use serverless services:** Prioritize building [serverless architectures](#) and [event-driven architecture](#) on AWS to build and run your applications. AWS offers multiple serverless technology services that inherently provide automatically optimized resource utilization and automated decommissioning (scale in and scale out). With serverless applications, resource utilization is automatically optimized and you never pay for over-provisioning.

Resources

Related documents:

- [Amazon EC2 Auto Scaling](#)
- [Getting Started with Amazon EC2 Auto Scaling](#)
- [Application Auto Scaling](#)
- [AWS Trusted Advisor](#)
- [Serverless on AWS](#)
- [Create Alarms to Stop, Terminate, Reboot, or Recover an Instance](#)
- [Adding terminate actions to Amazon CloudWatch alarms](#)

Related examples:

- [Scheduling automatic deletion of AWS CloudFormation stacks](#)
- [Well-Architected Labs – Decommission resources automatically \(Level 100\)](#)
- [Servian AWS Auto Cleanup](#)

COST04-BP05 Enforce data retention policies

Define data retention policies on supported resources to handle object deletion per your organizations' requirements. Identify and delete unnecessary or orphaned resources and objects that are no longer required.

Level of risk exposed if this best practice is not established: Medium

Use data retention policies and lifecycle policies to reduce the associated costs of the decommissioning process and storage costs for the identified resources. Defining your data retention policies and lifecycle policies to perform automated storage class migration and deletion will reduce the overall storage costs during its lifetime. You can use Amazon Data Lifecycle

Manager to automate the creation and deletion of Amazon Elastic Block Store snapshots and Amazon EBS-backed Amazon Machine Images (AMIs), and use Amazon S3 Intelligent-Tiering or an Amazon S3 lifecycle configuration to manage the lifecycle of your Amazon S3 objects. You can also implement custom code using the [API or SDK](#) to create lifecycle policies and policy rules for objects to be deleted automatically.

Implementation steps

- **Use Amazon Data Lifecycle Manager:** Use lifecycle policies on Amazon Data Lifecycle Manager to automate deletion of Amazon EBS snapshots and Amazon EBS-backed AMIs.
- **Set up lifecycle configuration on a bucket:** Use Amazon S3 lifecycle configuration on a bucket to define actions for Amazon S3 to take during an object's lifecycle, as well as deletion at the end of the object's lifecycle, based on your business requirements.

Resources

Related documents:

- [AWS Trusted Advisor](#)
- [Amazon Data Lifecycle Manager](#)
- [How to set lifecycle configuration on Amazon S3 bucket](#)

Related videos:

- [Automate Amazon EBS Snapshots with Amazon Data Lifecycle Manager](#)
- [Empty an Amazon S3 bucket using a lifecycle configuration rule](#)

Related examples:

- [Empty an Amazon S3 bucket using a lifecycle configuration rule](#)
- [Well-Architected Lab: Decommission resources automatically \(Level 100\)](#)

Cost-effective resources

Questions

- [COST 5. How do you evaluate cost when you select services?](#)

- [COST 6. How do you meet cost targets when you select resource type, size and number?](#)
- [COST 7. How do you use pricing models to reduce cost?](#)
- [COST 8. How do you plan for data transfer charges?](#)

COST 5. How do you evaluate cost when you select services?

Amazon EC2, Amazon EBS, and Amazon S3 are building-block AWS services. Managed services, such as Amazon RDS and Amazon DynamoDB, are higher level, or application level, AWS services. By selecting the appropriate building blocks and managed services, you can optimize this workload for cost. For example, using managed services, you can reduce or remove much of your administrative and operational overhead, freeing you to work on applications and business-related activities.

Best practices

- [COST05-BP01 Identify organization requirements for cost](#)
- [COST05-BP02 Analyze all components of the workload](#)
- [COST05-BP03 Perform a thorough analysis of each component](#)
- [COST05-BP04 Select software with cost-effective licensing](#)
- [COST05-BP05 Select components of this workload to optimize cost in line with organization priorities](#)
- [COST05-BP06 Perform cost analysis for different usage over time](#)

COST05-BP01 Identify organization requirements for cost

This best practice was updated with new guidance on December 6, 2023.

Work with team members to define the balance between cost optimization and other pillars, such as performance and reliability, for this workload.

Level of risk exposed if this best practice is not established: High

Implementation guidance

In most organizations, the information technology (IT) department is comprised of multiple small teams, each with its own agenda and focus area, that reflects the specialisies and skills of its

team members. You need to understand your organization's overall objectives, priorities, goals and how each department or project contributes to these objectives. Categorizing all essential resources, including personnel, equipment, technology, materials, and external services, is crucial for achieving organizational objectives and comprehensive budget planning. Adopting this systematic approach to cost identification and understanding is fundamental for establishing a realistic and robust cost plan for the organization.

When selecting services for your workload, it is key that you understand your organization priorities. Create a balance between cost optimization and other AWS Well-Architected Framework pillars, such as performance and reliability. This process should be conducted systematically and regularly to reflect changes in the organization's objectives, market conditions, and operational dynamics. A fully cost-optimized workload is the solution that is most aligned to your organization's requirements, not necessarily the lowest cost. Meet with all teams in your organization, such as product, business, technical, and finance to collect information. Evaluate the impact of tradeoffs between competing interests or alternative approaches to help make informed decisions when determining where to focus efforts or choosing a course of action.

For example, accelerating speed to market for new features may be emphasized over cost optimization, or you may choose a relational database for non-relational data to simplify the effort to migrate a system, rather than migrating to a database optimized for your data type and updating your application.

Implementation steps

- **Identify organization requirements for cost:** Meet with team members from your organization, including those in product management, application owners, development and operational teams, management, and financial roles. Prioritize the Well-Architected pillars for this workload and its components. The output should be a list of the pillars in order. You can also add a weight to each pillar to indicate how much additional focus it has, or how similar the focus is between two pillars.
- **Address the technical debt and document it:** During the workload review, address the technical debt. Document a backlog item to revisit the workload in the future, with the goal of refactoring or re-architecting to optimize it further. It's essential to clearly communicate the trade-offs that were made to other stakeholders.

Resources

Related best practices:

- [REL11-BP07 Architect your product to meet availability targets and uptime service level agreements \(SLAs\)](#)
- [OPS01-BP06 Evaluate tradeoffs](#)

Related documents:

- [AWS Total Cost of Ownership \(TCO\) Calculator](#)
- [Amazon S3 storage classes](#)
- [Cloud products](#)

COST05-BP02 Analyze all components of the workload

This best practice was updated with new guidance on December 6, 2023.

Verify every workload component is analyzed, regardless of current size or current costs. The review effort should reflect the potential benefit, such as current and projected costs.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Workload components, which are designed to deliver business value to the organization, may encompass various services. For each component, one might choose specific AWS Cloud services to address business needs. This selection could be influenced by factors such as familiarity with or prior experience using these services.

After identifying your organization's requirements (as mentioned in [COST05-BP01 Identify organization requirements for cost](#)), perform a thorough analysis on all components in your workload. Analyze each component considering current and projected costs and sizes. Consider the cost of analysis against any potential workload savings over its lifecycle. The effort expended on analyzing all components of this workload should correspond to the potential savings or improvements anticipated from optimizing that specific component. For example, if the cost of the proposed resource is \$10 per month, and under forecasted loads would not exceed \$15 per month, spending a day of effort to reduce costs by 50% (five dollars per month) could exceed the potential benefit over the life of the system. Using a faster and more efficient data-based estimation creates the best overall outcome for this component.

Workloads can change over time, and the right set of services may not be optimal if the workload architecture or usage changes. Analysis for selection of services must incorporate current and future workload states and usage levels. Implementing a service for future workload state or usage may reduce overall costs by reducing or removing the effort required to make future changes. For example, using Amazon EMR Serverless might be the appropriate choice initially. However, as consumption for that service increases, transitioning to Amazon EMR on Amazon EC2 could reduce costs for that component of the workload.

Strategic review of all workload components, irrespective of their present attributes, has the potential to bring about notable enhancements and financial savings over time. The effort invested in this review process should be deliberate, with careful consideration of the potential advantages that might be realized.

[AWS Cost Explorer](#) and the [AWS Cost and Usage Report \(CUR\)](#) can analyze the cost of a Proof of Concept (PoC) or running environment. You can also use [AWS Pricing Calculator](#) to estimate workload costs.

Implementation steps

- **List the workload components:** Build a list of your workload's components. This is used as verification to check that each component was analyzed. The effort spent should reflect the criticality to the workload as defined by your organization's priorities. Grouping together resources functionally improves efficiency (for example production database storage, if there are multiple databases).
- **Prioritize component list:** Take the component list and prioritize it in order of effort. This is typically in order of the cost of the component, from most expensive to least expensive, or the criticality as defined by your organization's priorities.
- **Perform the analysis:** For each component on the list, review the options and services available, and chose the option that aligns best with your organizational priorities.

Resources

Related documents:

- [AWS Pricing Calculator](#)
- [AWS Cost Explorer](#)
- [Amazon S3 storage classes](#)

- [Cloud products](#)

COST05-BP03 Perform a thorough analysis of each component

Look at overall cost to the organization of each component. Calculate the total cost of ownership by factoring in cost of operations and management, especially when using managed services by cloud provider. The review effort should reflect potential benefit (for example, time spent analyzing is proportional to component cost).

Level of risk exposed if this best practice is not established: High

Implementation guidance

Consider the time savings that will allow your team to focus on retiring technical debt, innovation, value-adding features and building what differentiates the business. For example, you might need to lift and shift (also known as rehost) your databases from your on-premises environment to the cloud as rapidly as possible and optimize later. It is worth exploring the possible savings attained by using managed services on AWS that may remove or reduce license costs. Managed services on AWS remove the operational and administrative burden of maintaining a service, such as patching or upgrading the OS, and allow you to focus on innovation and business.

Since managed services operate at cloud scale, they can offer a lower cost per transaction or service. You can make potential optimizations in order to achieve some tangible benefit, without changing the core architecture of the application. For example, you may be looking to reduce the amount of time you spend managing database instances by migrating to a database-as-a-service platform like [Amazon Relational Database Service \(Amazon RDS\)](#) or migrating your application to a fully managed platform like [AWS Elastic Beanstalk](#).

Usually, managed services have attributes that you can set to ensure sufficient capacity. You must set and monitor these attributes so that your excess capacity is kept to a minimum and performance is maximized. You can modify the attributes of AWS Managed Services using the AWS Management Console or AWS APIs and SDKs to align resource needs with changing demand. For example, you can increase or decrease the number of nodes on an Amazon EMR cluster (or an Amazon Redshift cluster) to scale out or in.

You can also pack multiple instances on an AWS resource to activate higher density usage. For example, you can provision multiple small databases on a single Amazon Relational Database Service (Amazon RDS) database instance. As usage grows, you can migrate one of the databases to a dedicated Amazon RDS database instance using a snapshot and restore process.

When provisioning workloads on managed services, you must understand the requirements of adjusting the service capacity. These requirements are typically time, effort, and any impact to normal workload operation. The provisioned resource must allow time for any changes to occur, provision the required overhead to allow this. The ongoing effort required to modify services can be reduced to virtually zero by using APIs and SDKs that are integrated with system and monitoring tools, such as Amazon CloudWatch.

[Amazon RDS](#), [Amazon Redshift](#), and [Amazon ElastiCache](#) provide a managed database service. [Amazon Athena](#), [Amazon EMR](#), and [Amazon OpenSearch Service](#) provide a managed analytics service.

[AMS](#) is a service that operates AWS infrastructure on behalf of enterprise customers and partners. It provides a secure and compliant environment that you can deploy your workloads onto. AMS uses enterprise cloud operating models with automation to allow you to meet your organization requirements, move into the cloud faster, and reduce your on-going management costs.

Implementation steps

- **Perform a thorough analysis:** Using the component list, work through each component from the highest priority to the lowest priority. For the higher priority and more costly components, perform additional analysis and assess all available options and their long term impact. For lower priority components, assess if changes in usage would change the priority of the component, and then perform an analysis of appropriate effort.
- **Compare managed and unmanaged resources:** Consider the operational cost for the resources you manage and compare them with AWS managed resources. For example, review your databases running on Amazon EC2 instances and compare with Amazon RDS options (an AWS managed service) or Amazon EMR compared to running Apache Spark on Amazon EC2. When moving from a self-managed workload to a AWS fully managed workload, research your options carefully. The three most important factors to consider are the [type of managed service](#) you want to use, the process you will use to [migrate your data](#) and understand the [AWS shared responsibility model](#).

Resources

Related documents:

- [AWS Total Cost of Ownership \(TCO\) Calculator](#)
- [Amazon S3 storage classes](#)

- [AWS Cloud products](#)
- [AWS Shared Responsibility Model](#)

Related videos:

- [Why move to a managed database?](#)
- [What is Amazon EMR and how can I use it for processing data?](#)

Related examples:

- [Why to move to a managed database](#)
- [Consolidate data from identical SQL Server databases into a single Amazon RDS for SQL Server database using AWS DMS](#)
- [Deliver data at scale to Amazon Managed Streaming for Apache Kafka \(Amazon MSK\)](#)
- [Migrate an ASP.NET web application to AWS Elastic Beanstalk](#)

COST05-BP04 Select software with cost-effective licensing

This best practice was updated with new guidance on December 6, 2023.

Open-source software eliminates software licensing costs, which can contribute significant costs to workloads. Where licensed software is required, avoid licenses bound to arbitrary attributes such as CPUs, look for licenses that are bound to output or outcomes. The cost of these licenses scales more closely to the benefit they provide.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Open source originated in the context of software development to indicate that the software complies with certain free distribution criteria. Open source software is composed of source code that anyone can inspect, modify, and enhance. Based on business requirements, skill of engineers, forecasted usage, or other technology dependencies, organizations can consider using open source software on AWS to minimize their license costs. In other words, the cost of software licenses can be reduced through the use of [open source software](#). This can have significant impact on workload costs as the size of the workload scales.

Measure the benefits of licensed software against the total cost to optimize your workload. Model any changes in licensing and how they would impact your workload costs. If a vendor changes the cost of your database license, investigate how that impacts the overall efficiency of your workload. Consider historical pricing announcements from your vendors for trends of licensing changes across their products. Licensing costs may also scale independently of throughput or usage, such as licenses that scale by hardware (CPU bound licenses). These licenses should be avoided because costs can rapidly increase without corresponding outcomes.

For instance, operating an Amazon EC2 instance in us-east-1 with a Linux operating system allows you to cut costs by approximately 45%, compared to running another Amazon EC2 instance that runs on Windows.

The [AWS Pricing Calculator](#) offers a comprehensive way to compare the costs of various resources with different license options, such as Amazon RDS instances and different database engines. Additionally, the AWS Cost Explorer provides an invaluable perspective for the costs of existing workloads, especially those that come with different licenses. For license management, [AWS License Manager](#) offers a streamlined method to oversee and handle software licenses. Customers can deploy and operationalize their preferred open source software in the AWS Cloud.

Implementation steps

- **Analyze license options:** Review the licensing terms of available software. Look for open source versions that have the required functionality, and whether the benefits of licensed software outweigh the cost. Favorable terms align the cost of the software to the benefits it provides.
- **Analyze the software provider:** Review any historical pricing or licensing changes from the vendor. Look for any changes that do not align to outcomes, such as punitive terms for running on specific vendors hardware or platforms. Additionally, look for how they perform audits, and penalties that could be imposed.

Resources

Related documents:

- [Open Source at AWS](#)
- [AWS Total Cost of Ownership \(TCO\) Calculator](#)
- [Amazon S3 storage classes](#)
- [Cloud products](#)

Related examples:

- [Open Source Blogs](#)
- [AWS Open Source Blogs](#)
- [Optimization and Licensing Assessment](#)

COST05-BP05 Select components of this workload to optimize cost in line with organization priorities

Factor in cost when selecting all components for your workload. This includes using application-level and managed services or serverless, containers, or event-driven architecture to reduce overall cost. Minimize license costs by using open-source software, software that does not have license fees, or alternatives to reduce spending.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Consider the cost of services and options when selecting all components. This includes using application level and managed services, such as [Amazon Relational Database Service](#) (Amazon RDS), [Amazon DynamoDB](#), [Amazon Simple Notification Service](#) (Amazon SNS), and [Amazon Simple Email Service](#) (Amazon SES) to reduce overall organization cost.

Use serverless and containers for compute, such as [AWS Lambda](#) and [Amazon Simple Storage Service](#) (Amazon S3) for static websites. Containerize your application if possible and use AWS Managed Container Services such as [Amazon Elastic Container Service](#) (Amazon ECS) or [Amazon Elastic Kubernetes Service](#) (Amazon EKS).

Minimize license costs by using open-source software, or software that does not have license fees (for example, Amazon Linux for compute workloads or migrate databases to Amazon Aurora).

You can use serverless or application-level services such as [Lambda](#), [Amazon Simple Queue Service \(Amazon SQS\)](#), [Amazon SNS](#), and [Amazon SES](#). These services remove the need for you to manage a resource and provide the function of code execution, queuing services, and message delivery. The other benefit is that they scale in performance and cost in line with usage, allowing efficient cost allocation and attribution.

Using [event-driven architecture](#) is also possible with serverless services. Event-driven architectures are push-based, so everything happens on demand as the event presents itself in the router.

This way, you're not paying for continuous polling to check for an event. This means less network bandwidth consumption, less CPU utilization, less idle fleet capacity, and fewer SSL/TLS handshakes.

For more information on serverless, see [Well-Architected Serverless Application lens whitepaper](#).

Implementation steps

- **Select each service to optimize cost:** Using your prioritized list and analysis, select each option that provides the best match with your organizational priorities. Instead of increasing the capacity to meet the demand, consider other options which may give you better performance with lower cost. For example, if you need to review expected traffic for your databases on AWS, consider either increasing the instance size or using Amazon ElastiCache services (Redis or Memcached) to provide cached mechanisms for your databases.
- **Evaluate event-driven architecture:** Using serverless architecture also allows you to build event-driven architecture for distributed microservice-based applications, which helps you build scalable, resilient, agile and cost-effective solutions.

Resources

Related documents:

- [AWS Total Cost of Ownership \(TCO\) Calculator](#)
- [AWS Serverless](#)
- [What is Event-Driven Architecture](#)
- [Amazon S3 storage classes](#)
- [Cloud products](#)
- [Amazon ElastiCache for Redis](#)

Related examples:

- [Getting started with event-driven architecture](#)
- [Event-driven architecture](#)
- [How Statsig runs 100x more cost-effectively using Amazon ElastiCache for Redis](#)
- [Best practices for working with AWS Lambda functions](#)

COST05-BP06 Perform cost analysis for different usage over time

Workloads can change over time. Some services or features are more cost effective at different usage levels. By performing the analysis on each component over time and at projected usage, the workload remains cost-effective over its lifetime.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

As AWS releases new services and features, the optimal services for your workload may change. Effort required should reflect potential benefits. Workload review frequency depends on your organization requirements. If it is a workload of significant cost, implementing new services sooner will maximize cost savings, so more frequent review can be advantageous. Another initiation for review is change in usage patterns. Significant changes in usage can indicate that alternate services would be more optimal.

If you need to move data into AWS Cloud, you can select any wide variety of services AWS offers and partner tools to help you migrate your data sets, whether they are files, databases, machine images, block volumes, or even tape backups. For example, to move a large amount of data to and from AWS or process data at the edge, you can use one of the AWS purpose-built devices to cost effectively move petabytes of data offline. Another example is for higher data transfer rates, a direct connect service may be cheaper than a VPN which provides the required consistent connectivity for your business.

Based on the cost analysis for different usage over time, review your scaling activity. Analyze the result to see if the scaling policy can be tuned to add instances with multiple instance types and purchase options. Review your settings to see if the minimum can be reduced to serve user requests but with a smaller fleet size, and add more resources to meet the expected high demand.

Perform cost analysis for different usage over time by discussing with stakeholders in your organization and use [AWS Cost Explorer's](#) forecast feature to predict the potential impact of service changes. Monitor usage level launches using AWS Budgets, CloudWatch billing alarms and AWS Cost Anomaly Detection to identify and implement the most cost-effective services sooner.

Implementation steps

- **Define predicted usage patterns:** Working with your organization, such as marketing and product owners, document what the expected and predicted usage patterns will be for the

workload. Discuss with business stakeholders about both historical and forecasted cost and usage increases and make sure increases align with business requirements. Identify calendar days, weeks, or months where you expect more users to use your AWS resources, which indicate that you should increase the capacity of the existing resources or adopt additional services to reduce the cost and increase performance.

- **Perform cost analysis at predicted usage:** Using the usage patterns defined, perform analysis at each of these points. The analysis effort should reflect the potential outcome. For example, if the change in usage is large, a thorough analysis should be performed to verify any costs and changes. In other words, when cost increases, usage should increase for business as well.

Resources

Related documents:

- [AWS Total Cost of Ownership \(TCO\) Calculator](#)
- [Amazon S3 storage classes](#)
- [Cloud products](#)
- [Amazon EC2 Auto Scaling](#)
- [Cloud Data Migration](#)
- [AWS Snow Family](#)

Related videos:

- [AWS OpsHub for Snow Family](#)

COST 6. How do you meet cost targets when you select resource type, size and number?

Verify that you choose the appropriate resource size and number of resources for the task at hand. You minimize waste by selecting the most cost effective type, size, and number.

Best practices

- [COST06-BP01 Perform cost modeling](#)
- [COST06-BP02 Select resource type, size, and number based on data](#)
- [COST06-BP03 Select resource type, size, and number automatically based on metrics](#)

COST06-BP01 Perform cost modeling

Identify organization requirements (such as business needs and existing commitments) and perform cost modeling (overall costs) of the workload and each of its components. Perform benchmark activities for the workload under different predicted loads and compare the costs. The modeling effort should reflect the potential benefit. For example, time spent is proportional to component cost.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Perform cost modelling for your workload and each of its components to understand the balance between resources, and find the correct size for each resource in the workload, given a specific level of performance. Understanding cost considerations can inform your organizational business case and decision-making process when evaluating the value realization outcomes for planned workload deployment.

Perform benchmark activities for the workload under different predicted loads and compare the costs. The modelling effort should reflect potential benefit; for example, time spent is proportional to component cost or predicted saving. For best practices, refer to the [Review section of the Performance Efficiency Pillar of the AWS Well-Architected Framework](#).

As an example, to create cost modeling for a workload consisting of compute resources, [AWS Compute Optimizer](#) can assist with cost modelling for running workloads. It provides right-sizing recommendations for compute resources based on historical usage. Make sure CloudWatch Agents are deployed to the Amazon EC2 instances to collect memory metrics which help you with more accurate recommendations within AWS Compute Optimizer. This is the ideal data source for compute resources because it is a free service that uses machine learning to make multiple recommendations depending on levels of risk.

There are [multiple services](#) you can use with custom logs as data sources for rightsizing operations for other services and workload components, such as [AWS Trusted Advisor](#), [Amazon CloudWatch](#) and [Amazon CloudWatch Logs](#). AWS Trusted Advisor checks resources and flags resources with low utilization which can help you right size your resources and create cost modelling.

The following are recommendations for cost modelling data and metrics:

- The monitoring must accurately reflect the user experience. Select the correct granularity for the time period and thoughtfully choose the maximum or 99th percentile instead of the average.

- Select the correct granularity for the time period of analysis that is required to cover any workload cycles. For example, if a two-week analysis is performed, you might be overlooking a monthly cycle of high utilization, which could lead to under-provisioning.
- Choose the right AWS services for your planned workload by considering your existing commitments, selected pricing models for other workloads, and ability to innovate faster and focus on your core business value.

Implementation steps

- **Perform cost modeling for resources:** Deploy the workload or a proof of concept into a separate account with the specific resource types and sizes to test. Run the workload with the test data and record the output results, along with the cost data for the time the test was run. Afterwards, redeploy the workload or change the resource types and sizes and run the test again. Include license fees of any products you may use with these resources and estimated operations (labor or engineer) costs for deploying and managing these resources while creating cost modeling. Consider cost modeling for a period (hourly, daily, monthly, yearly or three years).

Resources

Related documents:

- [AWS Auto Scaling](#)
- [Identifying Opportunities to Right Size](#)
- [Amazon CloudWatch features](#)
- [Cost Optimization: Amazon EC2 Right Sizing](#)
- [AWS Compute Optimizer](#)
- [AWS Pricing Calculator](#)

Related examples:

- [Perform a Data-Driven Cost Modelling](#)
- [Estimate the cost of planned AWS resource configurations](#)
- [Choose the right AWS tools](#)

COST06-BP02 Select resource type, size, and number based on data

This best practice was updated with new guidance on December 6, 2023.

Select resource size or type based on data about the workload and resource characteristics. For example, compute, memory, throughput, or write intensive. This selection is typically made using a previous (on-premises) version of the workload, using documentation, or using other sources of information about the workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Amazon EC2 provides a wide selection of instance types with different levels of CPU, memory, storage, and networking capacity to fit different use cases. These instance types feature different blends of CPU, memory, storage, and networking capabilities, giving you versatility when selecting the right resource combination for your projects. Every instance type comes in multiple sizes, so that you can adjust your resources based on your workload's demands. To determine which instance type you need, gather details about the system requirements of the application or software that you plan to run on your instance. These details should include the following:

- Operating system
- Number of CPU cores
- GPU cores
- Amount of system memory (RAM)
- Storage type and space
- Network bandwidth requirement

Identify the purpose of compute requirements and which instance is needed, and then explore the various Amazon EC2 instance families. Amazon offers the following instance type families:

- General Purpose
- Compute Optimized
- Memory Optimized
- Storage Optimized
- Accelerated Computing

- HPC Optimized

For a deeper understanding of the specific purposes and use cases that a particular Amazon EC2 instance family can fulfill, see [AWS Instance types](#).

System requirements gathering is critical for you to select the specific instance family and instance type that best serves your needs. Instance type names are comprised of the family name and the instance size. For example, the t2.micro instance is from the T2 family and is micro-sized.

Select resource size or type based on workload and resource characteristics (for example, compute, memory, throughput, or write intensive). This selection is typically made using cost modelling, a previous version of the workload (such as an on-premises version), using documentation, or using other sources of information about the workload (whitepapers or published solutions). Using AWS pricing calculators or cost management tools can assist in making informed decisions about instance types, sizes, and configurations.

Implementation steps

- **Select resources based on data:** Use your cost modeling data to select the anticipated workload usage level, and choose the specified resource type and size. Relying on the cost modeling data, determine the number of virtual CPUs, total memory (GiB), the local instance store volume (GB), Amazon EBS volumes, and the network performance level, taking into account the data transfer rate required for the instance. Always make selections based on detailed analysis and accurate data to optimize performance while managing costs effectively.

Resources

Related documents:

- [AWS Instance types](#)
- [AWS Auto Scaling](#)
- [Amazon CloudWatch features](#)
- [Cost Optimization: EC2 Right Sizing](#)

Related videos:

- [Selecting the right Amazon EC2 instance for your workloads](#)
- [Right size your service](#)

Related examples:

- [It just got easier to discover and compare Amazon EC2 instance types](#)

COST06-BP03 Select resource type, size, and number automatically based on metrics

Use metrics from the currently running workload to select the right size and type to optimize for cost. Appropriately provision throughput, sizing, and storage for compute, storage, data, and networking services. This can be done with a feedback loop such as automatic scaling or by custom code in the workload.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Create a feedback loop within the workload that uses active metrics from the running workload to make changes to that workload. You can use a managed service, such as [AWS Auto Scaling](#), which you configure to perform the right sizing operations for you. AWS also provides [APIs, SDKs](#), and features that allow resources to be modified with minimal effort. You can program a workload to stop-and-start an Amazon EC2 instance to allow a change of instance size or instance type. This provides the benefits of right-sizing while removing almost all the operational cost required to make the change.

Some AWS services have built in automatic type or size selection, such as [Amazon Simple Storage Service Intelligent-Tiering](#). Amazon S3 Intelligent-Tiering automatically moves your data between two access tiers, frequent access and infrequent access, based on your usage patterns.

Implementation steps

- **Increase your observability by configuring workload metrics:** Capture key metrics for the workload. These metrics provide an indication of the customer experience, such as workload output, and align to the differences between resource types and sizes, such as CPU and memory usage. For compute resource, analyze performance data to right size your Amazon EC2 instances. Identify idle instances and ones that are underutilized. Key metrics to look for are CPU usage and memory utilization (for example, 40% CPU utilization at 90% of the time as explained in [Rightsizing with AWS Compute Optimizer and Memory Utilization Enabled](#)). Identify instances with a maximum CPU usage and memory utilization of less than 40% over a four-week period. These are the instances to right size to reduce costs. For storage resources such as Amazon S3, you can use [Amazon S3 Storage Lens](#), which allows you to see 28 metrics across various categories at the bucket level, and 14 days of historical data in the dashboard by default. You can

filter your Amazon S3 Storage Lens dashboard by summary and cost optimization or events to analyze specific metrics.

- **View rightsizing recommendations:** Use the rightsizing recommendations in AWS Compute Optimizer and the Amazon EC2 rightsizing tool in the Cost Management console, or review AWS Trusted Advisor right-sizing your resources to make adjustments on your workload. It is important to use the [right tools](#) when right-sizing different resources and follow [right-sizing guidelines](#) whether it is an Amazon EC2 instance, AWS storage classes, or Amazon RDS instance types. For storage resources, you can use Amazon S3 Storage Lens, which gives you visibility into object storage usage, activity trends, and makes actionable recommendations to optimize costs and apply data protection best practices. Using the contextual recommendations that [Amazon S3 Storage Lens](#) derives from analysis of metrics across your organization, you can take immediate steps to optimize your storage.
- **Select resource type and size automatically based on metrics:** Using the workload metrics, manually or automatically select your workload resources. For compute resources, configuring AWS Auto Scaling or implementing code within your application can reduce the effort required if frequent changes are needed, and it can potentially implement changes sooner than a manual process. You can launch and automatically scale a fleet of On-Demand Instances and Spot Instances within a single Auto Scaling group. In addition to receiving discounts for using Spot Instances, you can use Reserved Instances or a Savings Plan to receive discounted rates of the regular On-Demand Instance pricing. All of these factors combined help you optimize your cost savings for Amazon EC2 instances and determine the desired scale and performance for your application. You can also use an [attribute-based instance type selection \(ABS\)](#) strategy in [Auto Scaling Groups \(ASG\)](#), which lets you express your instance requirements as a set of attributes, such as vCPU, memory, and storage. You can automatically use newer generation instance types when they are released and access a broader range of capacity with Amazon EC2 Spot Instances. Amazon EC2 Fleet and Amazon EC2 Auto Scaling select and launch instances that fit the specified attributes, removing the need to manually pick instance types. For storage resources, you can use the [Amazon S3 Intelligent Tiering](#) and [Amazon EFS Infrequent Access](#) features, which allow you to select storage classes automatically that deliver automatic storage cost savings when data access patterns change, without performance impact or operational overhead.

Resources

Related documents:

- [AWS Auto Scaling](#)

- [AWS Right-Sizing](#)
- [AWS Compute Optimizer](#)
- [Amazon CloudWatch features](#)
- [CloudWatch Getting Set Up](#)
- [CloudWatch Publishing Custom Metrics](#)
- [Getting Started with Amazon EC2 Auto Scaling](#)
- [Amazon S3 Storage Lens](#)
- [Amazon S3 Intelligent-Tiering](#)
- [Amazon EFS Infrequent Access](#)
- [Launch an Amazon EC2 Instance Using the SDK](#)

Related videos:

- [Right Size Your Services](#)

Related examples:

- [Attribute based Instance Type Selection for Auto Scaling for Amazon EC2 Fleet](#)
- [Optimizing Amazon Elastic Container Service for cost using scheduled scaling](#)
- [Predictive scaling with Amazon EC2 Auto Scaling](#)
- [Optimize Costs and Gain Visibility into Usage with Amazon S3 Storage Lens](#)
- [Well-Architected Labs: Rightsizing Recommendations \(Level 100\)](#)
- [Well-Architected Labs: Rightsizing with AWS Compute Optimizer and Memory Utilization Enabled \(Level 200\)](#)

COST 7. How do you use pricing models to reduce cost?

Use the pricing model that is most appropriate for your resources to minimize expense.

Best practices

- [COST07-BP01 Perform pricing model analysis](#)
- [COST07-BP02 Choose Regions based on cost](#)

- [COST07-BP03 Select third-party agreements with cost-efficient terms](#)
- [COST07-BP04 Implement pricing models for all components of this workload](#)
- [COST07-BP05 Perform pricing model analysis at the management account level](#)

COST07-BP01 Perform pricing model analysis

Analyze each component of the workload. Determine if the component and resources will be running for extended periods (for commitment discounts) or dynamic and short-running (for spot or on-demand). Perform an analysis on the workload using the recommendations in cost management tools and apply business rules to those recommendations to achieve high returns.

Level of risk exposed if this best practice is not established: High

Implementation guidance

AWS has multiple [pricing models](#) that allow you to pay for your resources in the most cost-effective way that suits your organization's needs and depending on product. Work with your teams to determine the most appropriate pricing model. Often your pricing model consists of a combination of multiple options, as determined by your availability

On-Demand Instances allow you pay for compute or database capacity by the hour or by the second (60 seconds minimum) depending on which instances you run, without long-term commitments or upfront payments.

Savings Plans are a flexible pricing model that offers low prices on Amazon EC2, Lambda, and AWS Fargate (Fargate) usage, in exchange for a commitment to a consistent amount of usage (measured in dollars per hour) over one year or three years terms.

Spot Instances are an Amazon EC2 pricing mechanism that allows you request spare compute capacity at discounted hourly rate (up to 90% off the on-demand price) without upfront commitment.

Reserved Instances allow you up to 75 percent discount by prepaying for capacity. For more details, see [Optimizing costs with reservations](#).

You might choose to include a Savings Plan for the resources associated with the production, quality, and development environments. Alternatively, because sandbox resources are only powered on when needed, you might choose an on-demand model for the resources in that environment. Use Amazon [Spot Instances](#) to reduce Amazon EC2 costs or use [Compute Savings](#)

[Plans](#) to reduce Amazon EC2, Fargate, and Lambda cost. The [AWS Cost Explorer](#) recommendations tool provides opportunities for commitment discounts with Saving plans.

If you have been purchasing [Reserved Instances](#) for Amazon EC2 in the past or have established cost allocation practices inside your organization, you can continue using Amazon EC2 Reserved Instances for the time being. However, we recommend working on a strategy to use Savings Plans in the future as a more flexible cost savings mechanism. You can refresh Savings Plans (SP) Recommendations in AWS Cost Management to generate new Savings Plans Recommendations at any time. Use Reserved Instances (RI) to reduce Amazon RDS, Amazon Redshift, Amazon ElastiCache, and Amazon OpenSearch Service costs. Saving Plans and Reserved Instances are available in three options: all upfront, partial upfront and no upfront payments. Use the recommendations provided in AWS Cost Explorer RI and SP purchase recommendations.

To find opportunities for Spot workloads, use an hourly view of your overall usage, and look for regular periods of changing usage or elasticity. You can use Spot Instances for various fault-tolerant and flexible applications. Examples include stateless web servers, API endpoints, big data and analytics applications, containerized workloads, CI/CD, and other flexible workloads.

Analyze your Amazon EC2 and Amazon RDS instances whether they can be turned off when you don't use (after hours and weekends). This approach will allow you to reduce costs by 70% or more compared to using them 24/7. If you have Amazon Redshift clusters that only need to be available at specific times, you can pause the cluster and later resume it. When the Amazon Redshift cluster or Amazon EC2 and Amazon RDS Instance is stopped, the compute billing halts and only the storage charge applies.

Note that [On-Demand Capacity reservations](#) (ODCR) are not a pricing discount. Capacity Reservations are charged at the equivalent On-Demand rate, whether you run instances in reserved capacity or not. They should be considered when you need to provide enough capacity for the resources you plan to run. ODCRs don't have to be tied to long-term commitments, as they can be cancelled when you no longer need them, but they can also benefit from the discounts that Savings Plans or Reserved Instances provide.

Implementation steps

- **Analyze workload elasticity:** Using the hourly granularity in Cost Explorer or a custom dashboard, analyze your workload's elasticity. Look for regular changes in the number of instances that are running. Short duration instances are candidates for Spot Instances or Spot Fleet.
 - [Well-Architected Lab: Cost Explorer](#)

- [Well-Architected Lab: Cost Visualization](#)
- **Review existing pricing contracts:** Review current contracts or commitments for long term needs. Analyze what you currently have and how much those commitments are in use. Leverage pre-existing contractual discounts or enterprise agreements. [Enterprise Agreements](#) give customers the option to tailor agreements that best suit their needs. For long term commitments, consider reserved pricing discounts, Reserved Instances or Savings Plans for the specific instance type, instance family, AWS Region, and Availability Zones.
- **Perform a commitment discount analysis:** Using Cost Explorer in your account, review the Savings Plans and Reserved Instance recommendations. To verify that you implement the correct recommendations with the required discounts and risk, follow the [Well-Architected labs](#).

Resources

Related documents:

- [Accessing Reserved Instance recommendations](#)
- [Instance purchasing options](#)
- [AWS Enterprise](#)

Related videos:

- [Save up to 90% and run production workloads on Spot](#)

Related examples:

- [Well-Architected Lab: Cost Explorer](#)
- [Well-Architected Lab: Cost Visualization](#)
- [Well-Architected Lab: Pricing Models](#)

COST07-BP02 Choose Regions based on cost

Resource pricing may be different in each Region. Identify Regional cost differences and only deploy in Regions with higher costs to meet latency, data residency and data sovereignty requirements. Factoring in Region cost helps you pay the lowest overall price for this workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

The [AWS Cloud Infrastructure](#) is global, hosted in [multiple locations world-wide](#), and built around AWS Regions, Availability Zones, Local Zones, AWS Outposts, and Wavelength Zones. A Region is a physical location in the world and each Region is a separate geographic area where AWS has multiple Availability Zones. Availability Zones which are multiple isolated locations within each Region consist of one or more discrete data centers, each with redundant power, networking, and connectivity.

Each AWS Region operates within local market conditions, and resource pricing is different in each Region due to differences in the cost of land, fiber, electricity, and taxes, for example. Choose a specific Region to operate a component of or your entire solution so that you can run at the lowest possible price globally. Use [AWS Calculator](#) to estimate the costs of your workload in various Regions by searching services by location type (Region, wave length zone and local zone) and Region.

When you architect your solutions, a best practice is to seek to place computing resources closer to users to provide lower latency and strong data sovereignty. Select the geographic location based on your business, data privacy, performance, and security requirements. For applications with global end users, use multiple locations.

Use Regions that provide lower prices for AWS services to deploy your workloads if you have no obligations in data privacy, security and business requirements. For example, if your default Region is Asia Pacific (Sydney) (ap-southwest-2), and if there are no restrictions (data privacy, security, for example) to use other Regions, deploying non-critical (development and test) Amazon EC2 instances in US East (N. Virginia) (us-east-1) will cost you less.

	<i>Compliance</i>	<i>Latency</i>	<i>Cost</i>	<i>Services / Features</i>
Region 1	✓	15 ms	\$\$	✓
Region 2	✓	20 ms	\$\$\$	X
Region 3	✓	80 ms	\$	✓
Region 4	✓	15 ms	\$\$	✓
Region 5	✓	20 ms	\$\$\$	X
Region 6	✓	15 ms	\$	✓
Region 7	✓	80 ms	\$	✓
Region 8	✓	15 ms	\$	X

Region feature matrix table

The preceding matrix table shows us that Region 6 is the best option for this given scenario because latency is low compared to other Regions, service is available, and it is the least expensive Region.

Implementation steps

- **Review AWS Region pricing:** Analyze the workload costs in the current Region. Starting with the highest costs by service and usage type, calculate the costs in other Regions that are available. If the forecasted saving outweighs the cost of moving the component or workload, migrate to the new Region.
- **Review requirements for multi-Region deployments:** Analyze your business requirements and obligations (data privacy, security, or performance) to find out if there are any restrictions for you to not to use multiple Regions. If there are no obligations to restrict you to use single Region, then use multiple Regions.
- **Analyze required data transfer:** Consider data transfer costs when selecting Regions. Keep your data close to your customer and close to the resources. Select less costly AWS Regions where data flows and where there is minimal data transfer. Depending on your business requirements for data transfer, you can use [Amazon CloudFront](#), [AWS PrivateLink](#), [AWS Direct Connect](#), and [AWS Virtual Private Network](#) to reduce your networking costs, improve performance, and enhance security.

Resources

Related documents:

- [Accessing Reserved Instance recommendations](#)
- [Amazon EC2 pricing](#)
- [Instance purchasing options](#)
- [Region Table](#)

Related videos:

- [Save up to 90% and run production workloads on Spot](#)

Related examples:

- [Overview of Data Transfer Costs for Common Architectures](#)
- [Cost Considerations for Global Deployments](#)
- [What to Consider when Selecting a Region for your Workloads](#)
- [Well-Architected Labs: Restrict service usage by Region \(Level 200\)](#)

COST07-BP03 Select third-party agreements with cost-efficient terms

This best practice was updated with new guidance on December 6, 2023.

Cost efficient agreements and terms ensure the cost of these services scales with the benefits they provide. Select agreements and pricing that scale when they provide additional benefits to your organization.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

There are multiple products on the market that can help you manage costs in your cloud environments. They may have some differences in terms of features that depend on customer requirements, such as some focusing on cost governance or cost visibility and others on cost optimization. One key factor for effective cost optimization and governance is using the right tool with necessary features and the right pricing model. These products have different pricing models. Some charge you a certain percentage of your monthly bill, while others charge a percentage of your realized savings. Ideally, you should pay only for what you need.

When you use third-party solutions or services in the cloud, it's important that the pricing structures are aligned to your desired outcomes. Pricing should scale with the outcomes and value it provides. For example, in software that takes a percentage of savings it provides, the more you save (outcome), the more it charges. License agreements where you pay more as your expenses increase might not always be in your best interest for optimizing costs. However, if the vendor offers clear benefits for all parts of your bill, this scaling fee might be justified.

For example, a solution that provides recommendations for Amazon EC2 and charges a percentage of your entire bill can become more expensive if you use other services that provide no benefit.

Another example is a managed service that is charged at a percentage of the cost of managed resources. A larger instance size may not necessarily require more management effort, but can be charged more. Verify that these service pricing arrangements include a cost optimization program or features in their service to drive efficiency.

Customers may find these products on the market more advanced or easier to use. You need to consider the cost of these products and think about potential cost optimization outcomes in the long term.

Implementation steps

- **Analyze third-party agreements and terms:** Review the pricing in third party agreements. Perform modeling for different levels of your usage, and factor in new costs such as new service usage, or increases in current services due to workload growth. Decide if the additional costs provide the required benefits to your business.

Resources

Related documents:

- [Accessing Reserved Instance recommendations](#)
- [Instance purchasing options](#)

Related videos:

- [Save up to 90% and run production workloads on Spot](#)

COST07-BP04 Implement pricing models for all components of this workload

This best practice was updated with new guidance on December 6, 2023.

Permanently running resources should utilize reserved capacity such as Savings Plans or Reserved Instances. Short-term capacity is configured to use Spot Instances, or Spot Fleet. On-Demand Instances are only used for short-term workloads that cannot be interrupted and do not run long enough for reserved capacity, between 25% to 75% of the period, depending on the resource type.

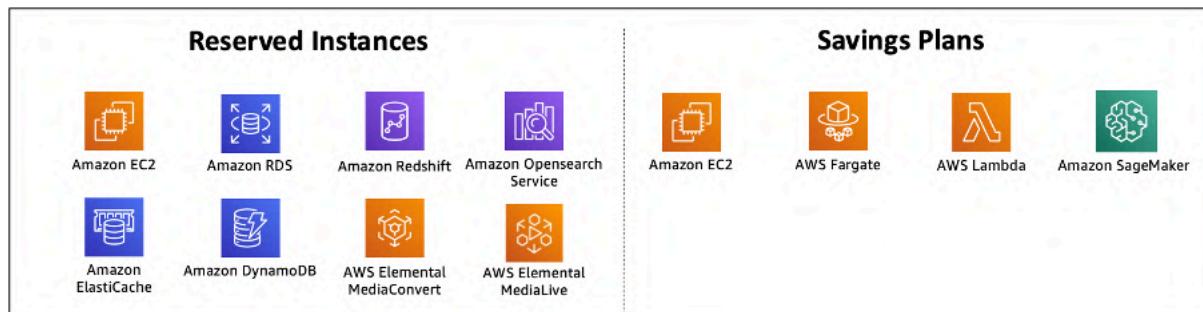
Level of risk exposed if this best practice is not established: Low

Implementation guidance

To improve cost efficiency, AWS provides multiple commitment recommendations based on your past usage. You can use these recommendations to understand what you can save, and how the commitment will be used. You can use these services as On-Demand, Spot, or make a commitment for a certain period of time and reduce your on-demand costs with Reserved Instances (RIs) and Savings Plans (SPs). You need to understand not only each workload components and multiple AWS services, but also commitment discounts, purchase options, and Spot Instances for these services to optimize your workload.

Consider the requirements of your workload's components, and understand the different pricing models for these services. Define the availability requirement of these components. Determine if there are multiple independent resources that perform the function in the workload, and what the workload requirements are over time. Compare the cost of the resources using the default On-Demand pricing model and other applicable models. Factor in any potential changes in resources or workload components.

For example, let's look at this Web Application Architecture on AWS. This sample workload consists of multiple AWS services, such as Amazon Route 53, AWS WAF, Amazon CloudFront, Amazon EC2 instances, Amazon RDS instances, Load Balancers, Amazon S3 storage, and Amazon Elastic File System (Amazon EFS). You need to review each of these services, and identify potential cost saving opportunities with different pricing models. Some of them may be eligible for RIs or SPs, while some of them may be only available by on-demand. As the following image shows, some of the AWS services can be committed using RIs or SPs.



AWS services committed using Reserved Instances and Savings Plans

Implementation steps

- Implement pricing models:** Using your analysis results, purchase Savings Plans, Reserved Instances, or implement Spot Instances. If it is your first commitment purchase, choose the top five or ten recommendations in the list, then monitor and analyze the results over the next

month or two. AWS Cost Management Console guides you through the process. Review the RI or SP recommendations from the console, customize the recommendations (type, payment, and term), and review hourly commitment (for example \$20 per hour), and then add to cart. Discounts apply automatically to eligible usage. Purchase a small amount of commitment discounts in regular cycles (for example every 2 weeks or monthly). Implement Spot Instances for workloads that can be interrupted or are stateless. Finally, select on-demand Amazon EC2 instances and allocate resources for the remaining requirements.

- **Workload review cycle:** Implement a review cycle for the workload that specifically analyzes pricing model coverage. Once the workload has the required coverage, purchase additional commitment discounts partially (every few months), or as your organization usage changes.

Resources

Related documents:

- [Understanding your Savings Plans recommendations](#)
- [Accessing Reserved Instance recommendations](#)
- [How to Purchase Reserved Instances](#)
- [Instance purchasing options](#)
- [Spot Instances](#)
- [Reservation models for other AWS services](#)
- [Savings Plans Supported Services](#)

Related videos:

- [Save up to 90% and run production workloads on Spot](#)

Related examples:

- [What should you consider before purchasing Savings Plans?](#)
- [How can I use Cost Explorer to analyze my spending and usage?](#)

COST07-BP05 Perform pricing model analysis at the management account level

Check billing and cost management tools and see recommended discounts with commitments and reservations to perform regular analysis at the management account level.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Performing regular cost modeling helps you implement opportunities to optimize across multiple workloads. For example, if multiple workloads use On-Demand Instances at an aggregate level, the risk of change is lower, and implementing a commitment-based discount can achieve a lower overall cost. It is recommended to perform analysis in regular cycles of two weeks to one month. This allows you to make small adjustment purchases, so the coverage of your pricing models continues to evolve with your changing workloads and their components.

Use the [AWS Cost Explorer](#) recommendations tool to find opportunities for commitment discounts in your management account. Recommendations at the management account level are calculated considering usage across all of the accounts in your AWS organization that have Reserve Instances (RI) or Savings Plans (SP). They're also calculated when discount sharing is activated to recommend a commitment that maximizes savings across accounts.

While purchasing at the management account level optimizes for max savings in many cases, there may be situations where you might consider purchasing SPs at the linked account level, like when you want the discounts to apply first to usage in that particular linked account. Member account recommendations are calculated at the individual account level, to maximize savings for each isolated account. If your account owns both RI and SP commitments, they will be applied in this order:

1. Zonal RI
2. Standard RI
3. Convertible RI
4. Instance Savings Plan
5. Compute Savings Plan

If you purchase an SP at the management account level, the savings will be applied based on highest to lowest discount percentage. SPs at the management account level look across all linked accounts and apply the savings wherever the discount will be the highest. If you wish to restrict

where the savings are applied, you can purchase a Savings Plan at the linked account level and any time that account is running eligible compute services, the discount will be applied there first. When the account is not running eligible compute services, the discount will be shared across the other linked accounts under the same management account. Discount sharing is turned on by default, but can be turned off if needed.

In a Consolidated Billing Family, Savings Plans are applied first to the owner account's usage, and then to other accounts' usage. This occurs only if you have sharing enabled. Your Savings Plans are applied to your highest savings percentage first. If there are multiple usages with equal savings percentages, Savings Plans are applied to the first usage with the lowest Savings Plans rate. Savings Plans continue to apply until there are no more remaining uses or your commitment is exhausted. Any remaining usage is charged at the On-Demand rates. You can refresh Savings Plans Recommendations in AWS Cost Management to generate new Savings Plans Recommendations at any time.

After analyzing flexibility of instances, you can commit by following recommendations. Create cost modeling by analyzing the workload's short-term costs with potential different resource options, analyzing AWS pricing models, and aligning them with your business requirements to find out total cost of ownership and [cost optimization](#) opportunities.

Implementation steps

Perform a commitment discount analysis: Use Cost Explorer in your account review the Savings Plans and Reserved Instance recommendations. Make sure you understand Saving Plan recommendations, and estimate your monthly spend and monthly savings. Review recommendations at the management account level, which are calculated considering usage across all of the member accounts in your AWS organization that have RI or Savings Plans discount sharing enabled for maximum savings across accounts. You can verify that you implemented the correct recommendations with the required discounts and risk by following the Well-Architected labs.

Resources

Related documents:

- [How does AWS pricing work?](#)
- [Instance purchasing options](#)
- [Saving Plan Overview](#)

- [Saving Plan recommendations](#)
- [Accessing Reserved Instance recommendations](#)
- [Understanding your Saving Plans recommendation](#)
- [How Savings Plans apply to your AWS usage](#)
- [Saving Plans with Consolidated Billing](#)
- [Turning on shared reserved instances and Savings Plans discounts](#)

Related videos:

- [Save up to 90% and run production workloads on Spot](#)

Related examples:

- [AWS Well-Architected Lab: Pricing Models \(Level 200\)](#)
- [AWS Well-Architected Labs: Pricing Model Analysis \(Level 200\)](#)
- [What should I consider before purchasing a Savings Plan?](#)
- [How can I use rolling Savings Plans to reduce commitment risk?](#)
- [When to Use Spot Instances](#)

COST 8. How do you plan for data transfer charges?

Verify that you plan and monitor data transfer charges so that you can make architectural decisions to minimize costs. A small yet effective architectural change can drastically reduce your operational costs over time.

Best practices

- [COST08-BP01 Perform data transfer modeling](#)
- [COST08-BP02 Select components to optimize data transfer cost](#)
- [COST08-BP03 Implement services to reduce data transfer costs](#)

COST08-BP01 Perform data transfer modeling

This best practice was updated with new guidance on December 6, 2023.

Gather organization requirements and perform data transfer modeling of the workload and each of its components. This identifies the lowest cost point for its current data transfer requirements.

Level of risk exposed if this best practice is not established: High

Implementation guidance

When designing a solution in the cloud, data transfer fees are usually neglected due to habits of designing architecture using on-premises data centers or lack of knowledge. Data transfer charges in AWS are determined by the source, destination, and volume of traffic. Factoring in these fees during the design phase can lead to cost savings. Understanding where the data transfer occurs in your workload, the cost of the transfer, and its associated benefit is very important to accurately estimate total cost of ownership (TCO). This allows you to make an informed decision to modify or accept the architectural decision. For example, you may have a Multi-Availability Zone configuration where you replicate data between the Availability Zones.

You model the components of services which transfer the data in your workload, and decide that this is an acceptable cost (similar to paying for compute and storage in both Availability Zones) to achieve the required reliability and resilience. Model the costs over different usage levels. Workload usage can change over time, and different services may be more cost effective at different levels.

While modelling your data transfer, think about how much data is ingested and where that data comes from. Additionally, consider how much data is processed and how much storage or compute capacity is needed. During modelling, follow networking best practices for your workload architecture to optimize your potential data transfer costs.

The AWS Pricing Calculator can help you see estimated costs for specific AWS services and expected data transfer. If you have a workload already running (for test purposes or in a pre-production environment), use [AWS Cost Explorer](#) or [AWS Cost and Usage Report](#) (CUR) to understand and model your data transfer costs. Configure a proof of concept (PoC) or test your workload, and run a test with a realistic simulated load. You can model your costs at different workload demands.

Implementation steps

- **Identify requirements:** What is the primary goal and business requirements for the planned data transfer between source and destination? What is the expected business outcome at the end? Gather business requirements and define expected outcome.
- **Identify source and destination:** What is the data source and destination for the data transfer, such as within AWS Regions, to AWS services, or out to the internet?

- [Data transfer within an AWS Region](#)
- [Data transfer between AWS Regions](#)
- [Data transfer out to the internet](#)
- **Identify data classifications:** What is the data classification for this data transfer? What kind of data is it? How big is the data? How frequently must data be transferred? Is data sensitive?
- **Identify AWS services or tools to use:** Which AWS services are used for this data transfer? Is it possible to use an already-provisioned service for another workload?
- **Calculate data transfer costs:** Use [AWS Pricing](#) the data transfer modeling you created previously to calculate the data transfer costs for the workload. Calculate the data transfer costs at different usage levels, for both increases and reductions in workload usage. Where there are multiple options for the workload architecture, calculate the cost for each option for comparison.
- **Link costs to outcomes:** For each data transfer cost incurred, specify the outcome that it achieves for the workload. If it is transfer between components, it may be for decoupling, if it is between Availability Zones it may be for redundancy.
- **Create data transfer modeling:** After gathering all information, create a conceptual base data transfer modeling for multiple use cases and different workloads.

Resources

Related documents:

- [AWS caching solutions](#)
- [AWS Pricing](#)
- [Amazon EC2 Pricing](#)
- [Amazon VPC pricing](#)
- [Understanding data transfer charges](#)

Related videos:

- [Monitoring and Optimizing Your Data Transfer Costs](#)
- [S3 Transfer Acceleration](#)

Related examples:

- [Overview of Data Transfer Costs for Common Architectures](#)
- [AWS Prescriptive Guidance for Networking](#)

COST08-BP02 Select components to optimize data transfer cost

This best practice was updated with new guidance on December 6, 2023.

All components are selected, and architecture is designed to reduce data transfer costs. This includes using components such as wide-area-network (WAN) optimization and Multi-Availability Zone (AZ) configurations

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Architecting for data transfer minimizes data transfer costs. This may involve using content delivery networks to locate data closer to users, or using dedicated network links from your premises to AWS. You can also use WAN optimization and application optimization to reduce the amount of data that is transferred between components.

When transferring data to or within the AWS Cloud, it is essential to know the destination based on varied use cases, the nature of the data, and the available network resources in order to select the right AWS services to optimize data transfer. AWS offers a range of data transfer services tailored for diverse data migration requirements. Select the right [data storage](#) and [data transfer](#) options based on the business needs within your organization.

When planning or reviewing your workload architecture, consider the following:

- **Use VPC endpoints within AWS:** VPC endpoints allow for private connections between your VPC and supported AWS services. This allows you to avoid using the public internet, which can lead to data transfer costs.
- **Use a NAT gateway:** Use a [NAT gateway](#) so that instances in a private subnet can connect to the internet or to the services outside your VPC. Check whether the resources behind the NAT gateway that send the most traffic are in the same Availability Zone as the NAT gateway. If they are not, create new NAT gateways in the same Availability Zone as the resource to reduce cross-AZ data transfer charges.

- **Use AWS Direct Connect** AWS Direct Connect bypasses the public internet and establishes a direct, private connection between your on-premises network and AWS. This can be more cost-effective and consistent than transferring large volumes of data over the internet.
- **Avoid transferring data across Regional boundaries:** Data transfers between AWS Regions (from one Region to another) typically incur charges. It should be a very thoughtful decision to pursue a multi-Region path. For more detail, see [Multi-Region scenarios](#).
- **Monitor data transfer:** Use Amazon CloudWatch and [VPC flow logs](#) to capture details about your data transfer and network usage. Analyze captured network traffic information in your VPCs, such as IP address or range going to and from network interfaces.
- **Analyze your network usage:** Use metering and reporting tools such as AWS Cost Explorer, CUDOS Dashboards, or CloudWatch to understand data transfer cost of your workload.

Implementation steps

- **Select components for data transfer:** Using the data transfer modeling explained in [COST08-BP01 Perform data transfer modeling](#), focus on where the largest data transfer costs are or where they would be if the workload usage changes. Look for alternative architectures or additional components that remove or reduce the need for data transfer (or lower its cost).

Resources

Related best practices:

- [COST08-BP01 Perform data transfer modeling](#)
- [COST08-BP03 Implement services to reduce data transfer costs](#)

Related documents:

- [Cloud Data Migration](#)
- [AWS caching solutions](#)
- [Deliver content faster with Amazon CloudFront](#)

Related examples:

- [Overview of Data Transfer Costs for Common Architectures](#)
- [AWS Network Optimization Tips](#)

- [Optimize performance and reduce costs for network analytics with VPC Flow Logs in Apache Parquet format](#)

COST08-BP03 Implement services to reduce data transfer costs

Implement services to reduce data transfer. For example, use edge locations or content delivery networks (CDN) to deliver content to end users, build caching layers in front of your application servers or databases, and use dedicated network connections instead of VPN for connectivity to the cloud.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

There are various AWS services that can help you to optimize your network data transfer usage. Depending on your workload components, type, and cloud architecture, these services can assist you in compression, caching, and sharing and distribution of your traffic on the cloud.

- [Amazon CloudFront](#) is a global content delivery network that delivers data with low latency and high transfer speeds. It caches data at edge locations across the world, which reduces the load on your resources. By using CloudFront, you can reduce the administrative effort in delivering content to large numbers of users globally with minimum latency. The [security savings bundle](#) can help you to save up to 30% on your CloudFront usage if you plan to grow your usage over time.
- [AWS Direct Connect](#) allows you to establish a dedicated network connection to AWS. This can reduce network costs, increase bandwidth, and provide a more consistent network experience than internet-based connections.
- [AWS VPN](#) allows you to establish a secure and private connection between your private network and the AWS global network. It is ideal for small offices or business partners because it provides simplified connectivity, and it is a fully managed and elastic service.
- [VPC Endpoints](#) allow connectivity between AWS services over private networking and can be used to reduce public data transfer and [NAT gateway](#) costs. [Gateway VPC endpoints](#) have no hourly charges, and support Amazon S3 and Amazon DynamoDB. [Interface VPC endpoints](#) are provided by [AWS PrivateLink](#) and have an hourly fee and per-GB usage cost.
- [NAT gateways](#) provide built-in scaling and management for reducing costs as opposed to a standalone NAT instance. Place NAT gateways in the same Availability Zones as high traffic instances and consider using VPC endpoints for the instances that need to access Amazon DynamoDB or Amazon S3 to reduce the data transfer and processing costs.

- Use [AWS Snow Family](#) devices which have computing resources to collect and process data at the edge. AWS Snow Family devices ([Snowcone](#), [Snowball](#) and [Snowmobile](#)) allow you to move petabytes of data to the AWS Cloud cost effectively and offline.

Implementation steps

- **Implement services:** Select applicable AWS network services based on your service workload type using the data transfer modeling and reviewing VPC Flow Logs. Look at where the largest costs and highest volume flows are. Review the AWS services and assess whether there is a service that reduces or removes the transfer, specifically networking and content delivery. Also look for caching services where there is repeated access to data or large amounts of data.

Resources

Related documents:

- [AWS Direct Connect](#)
- [AWS Explore Our Products](#)
- [AWS caching solutions](#)
- [Amazon CloudFront](#)
- [AWS Snow Family](#)
- [Amazon CloudFront Security Savings Bundle](#)

Related videos:

- [Monitoring and Optimizing Your Data Transfer Costs](#)
- [AWS Cost Optimization Series: CloudFront](#)
- [How can I reduce data transfer charges for my NAT gateway?](#)

Related examples:

- [How-to chargeback shared services: An AWS Transit Gateway example](#)
- [Understand AWS data transfer details in depth from cost and usage report using Athena query and QuickSight](#)
- [Overview of Data Transfer Costs for Common Architectures](#)

- [Using AWS Cost Explorer to analyze data transfer costs](#)
- [Cost-Optimizing your AWS architectures by utilizing Amazon CloudFront features](#)
- [How can I reduce data transfer charges for my NAT gateway?](#)

Manage demand and supply resources

Question

- [COST 9. How do you manage demand, and supply resources?](#)

COST 9. How do you manage demand, and supply resources?

For a workload that has balanced spend and performance, verify that everything you pay for is used and avoid significantly underutilizing instances. A skewed utilization metric in either direction has an adverse impact on your organization, in either operational costs (degraded performance due to over-utilization), or wasted AWS expenditures (due to over-provisioning).

Best practices

- [COST09-BP01 Perform an analysis on the workload demand](#)
- [COST09-BP02 Implement a buffer or throttle to manage demand](#)
- [COST09-BP03 Supply resources dynamically](#)

COST09-BP01 Perform an analysis on the workload demand

Analyze the demand of the workload over time. Verify that the analysis covers seasonal trends and accurately represents operating conditions over the full workload lifetime. Analysis effort should reflect the potential benefit, for example, time spent is proportional to the workload cost.

Level of risk exposed if this best practice is not established: High

Implementation guidance

Analyzing workload demand for cloud computing involves understanding the patterns and characteristics of computing tasks that are initiated in the cloud environment. This analysis helps users optimize resource allocation, manage costs, and verify that performance meets required levels.

Know the requirements of the workload. Your organization's requirements should indicate the workload response times for requests. The response time can be used to determine if the demand is managed, or if the supply of resources should change to meet the demand.

The analysis should include the predictability and repeatability of the demand, the rate of change in demand, and the amount of change in demand. Perform the analysis over a long enough period to incorporate any seasonal variance, such as end-of-month processing or holiday peaks.

Analysis effort should reflect the potential benefits of implementing scaling. Look at the expected total cost of the component and any increases or decreases in usage and cost over the workload's lifetime.

The following are some key aspects to consider when performing workload demand analysis for cloud computing:

1. **Resource utilization and performance metrics:** Analyze how AWS resources are being used over time. Determine peak and off-peak usage patterns to optimize resource allocation and scaling strategies. Monitor performance metrics such as response times, latency, throughput, and error rates. These metrics help assess the overall health and efficiency of the cloud infrastructure.
2. **User and application scaling behaviour:** Understand user behavior and how it affects workload demand. Examining the patterns of user traffic assists in enhancing the delivery of content and the responsiveness of applications. Analyze how workloads scale with increasing demand. Determine whether auto-scaling parameters are configured correctly and effectively for handling load fluctuations.
3. **Workload types:** Identify the different types of workloads running in the cloud, such as batch processing, real-time data processing, web applications, databases, or machine learning. Each type of workload may have different resource requirements and performance profiles.
4. **Service-level agreements (SLAs):** Compare actual performance with SLAs to ensure compliance and identify areas that need improvement.

You can use [Amazon CloudWatch](#) to collect and track metrics, monitor log files, set alarms, and automatically react to changes in your AWS resources. You can also use Amazon CloudWatch to gain system-wide visibility into resource utilization, application performance, and operational health.

With [AWS Trusted Advisor](#), you can provision your resources following best practices to improve system performance and reliability, increase security, and look for opportunities to save money.

You can also turn off non-production instances and use Amazon CloudWatch and Auto Scaling to match increases or reductions in demand.

Finally, you can use [AWS Cost Explorer](#) or [Amazon QuickSight](#) with the AWS Cost and Usage Report (CUR) file or your application logs to perform advanced analysis of workload demand.

Overall, a comprehensive workload demand analysis allows organizations to make informed decisions about resource provisioning, scaling, and optimization, leading to better performance, cost efficiency, and user satisfaction.

Implementation steps

- **Analyze existing workload data:** Analyze data from the existing workload, previous versions of the workload, or predicted usage patterns. Use Amazon CloudWatch, log files and monitoring data to gain insight on how workload was used. Analyze a full cycle of the workload, and collect data for any seasonal changes such as end-of-month or end-of-year events. The effort reflected in the analysis should reflect the workload characteristics. The largest effort should be placed on high-value workloads that have the largest changes in demand. The least effort should be placed on low-value workloads that have minimal changes in demand.
- **Forecast outside influence:** Meet with team members from across the organization that can influence or change the demand in the workload. Common teams would be sales, marketing, or business development. Work with them to know the cycles they operate within, and if there are any events that would change the demand of the workload. Forecast the workload demand with this data.

Resources

Related documents:

- [Amazon CloudWatch](#)
- [AWS Trusted Advisor](#)
- [AWS X-Ray](#)
- [AWS Auto Scaling](#)
- [AWS Instance Scheduler](#)
- [Getting started with Amazon SQS](#)
- [AWS Cost Explorer](#)
- [Amazon QuickSight](#)

Related videos:

Related examples:

- [Monitor, Track and Analyze for cost optimization](#)
- [Searching and analyzing logs in CloudWatch](#)

COST09-BP02 Implement a buffer or throttle to manage demand

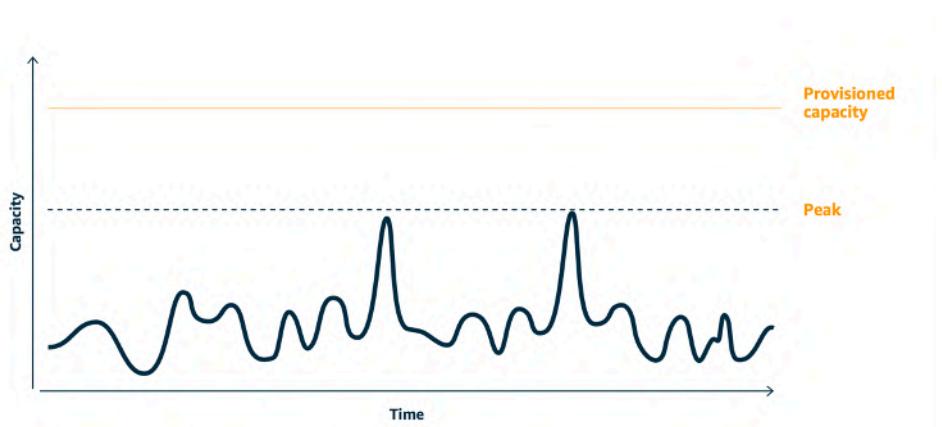
This best practice was updated with new guidance on December 6, 2023.

Buffering and throttling modify the demand on your workload, smoothing out any peaks. Implement throttling when your clients perform retries. Implement buffering to store the request and defer processing until a later time. Verify that your throttles and buffers are designed so clients receive a response in the required time.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Implementing a buffer or throttle is crucial in cloud computing in order to manage demand and reduce the provisioned capacity required for your workload. For optimal performance, it's essential to gauge the total demand, including peaks, the pace of change in requests, and the necessary response time. When clients have the ability to resend their requests, it becomes practical to apply throttling. Conversely, for clients lacking retry functionalities, the ideal approach is implementing a buffer solution. Such buffers streamline the influx of requests and optimize the interaction of applications with varied operational speeds.



Demand curve with two distinct peaks that require high provisioned capacity

Assume a workload with the demand curve shown in preceding image. This workload has two peaks, and to handle those peaks, the resource capacity as shown by orange line is provisioned. The resources and energy used for this workload are not indicated by the area under the demand curve, but the area under the provisioned capacity line, as provisioned capacity is needed to handle those two peaks. Flattening the workload demand curve can help you to reduce the provisioned capacity for a workload and reduce its environmental impact. To smooth out the peak, consider to implement throttling or buffering solution.

To understand them better, let's explore throttling and buffering.

Throttling: If the source of the demand has retry capability, then you can implement throttling. Throttling tells the source that if it cannot service the request at the current time, it should try again later. The source waits for a period of time, and then retries the request. Implementing throttling has the advantage of limiting the maximum amount of resources and costs of the workload. In AWS, you can use [Amazon API Gateway](#) to implement throttling.

Buffer based: A buffer-based approach uses *producers* (components that send messages to the queue), *consumers* (components that receive messages from the queue), and a *queue* (which holds messages) to store the messages. Messages are read by consumers and processed, allowing the messages to run at the rate that meets the consumers' business requirements. By using a buffer-centric methodology, messages from producers are housed in queues or streams, ready to be accessed by consumers at a pace that aligns with their operational demands.

In AWS, you can choose from multiple services to implement a buffering approach. [Amazon Simple Queue Service\(Amazon SQS\)](#) is a managed service that provides queues that allow a single consumer to read individual messages. [Amazon Kinesis](#) provides a stream that allows many consumers to read the same messages.

Buffering and throttling can smooth out any peaks by modifying the demand on your workload. Use throttling when clients retry actions and use buffering to hold the request and process it later. When working with a buffer-based approach, architect your workload to service the request in the required time, verify that you are able to handle duplicate requests for work. Analyze the overall demand, rate of change, and required response time to right size the throttle or buffer required.

Implementation steps

- **Analyze the client requirements:** Analyze the client requests to determine if they are capable of performing retries. For clients that cannot perform retries, buffers need to be implemented.

Analyze the overall demand, rate of change, and required response time to determine the size of throttle or buffer required.

- **Implement a buffer or throttle:** Implement a buffer or throttle in the workload. A queue such as Amazon Simple Queue Service (Amazon SQS) can provide a buffer to your workload components. Amazon API Gateway can provide throttling for your workload components.

Resources

Related best practices:

- [SUS02-BP06 Implement buffering or throttling to flatten the demand curve](#)
- [REL05-BP02 Throttle requests](#)

Related documents:

- [AWS Auto Scaling](#)
- [AWS Instance Scheduler](#)
- [Amazon API Gateway](#)
- [Amazon Simple Queue Service](#)
- [Getting started with Amazon SQS](#)
- [Amazon Kinesis](#)

Related videos:

- [Choosing the Right Messaging Service for Your Distributed App](#)

Related examples:

- [Managing and monitoring API throttling in your workloads](#)
- [Throttling a tiered, multi-tenant REST API at scale using API Gateway](#)
- [Enabling Tiering and Throttling in a Multi-Tenant Amazon EKS SaaS Solution Using Amazon API Gateway](#)
- [Application integration Using Queues and Messages](#)

COST09-BP03 Supply resources dynamically

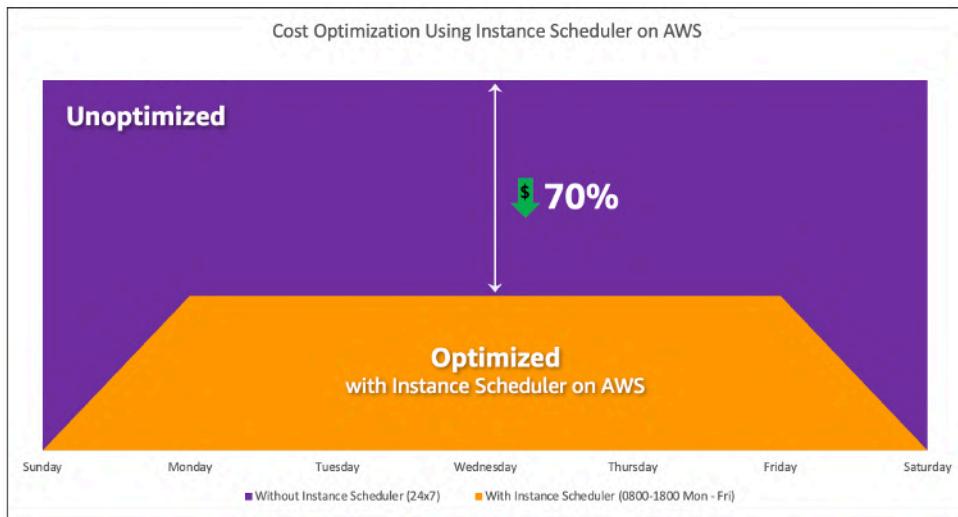
Resources are provisioned in a planned manner. This can be demand-based, such as through automatic scaling, or time-based, where demand is predictable and resources are provided based on time. These methods result in the least amount of over-provisioning or under-provisioning.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

There are several ways for AWS customers to increase the resources available to their applications and supply resources to meet the demand. One of these options is to use AWS Instance Scheduler, which automates the starting and stopping of Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Relational Database Service (Amazon RDS) instances. The other option is to use AWS Auto Scaling, which allows you to automatically scale your computing resources based on the demand of your application or service. Supplying resources based on demand will allow you to pay for the resources you use only, reduce cost by launching resources when they are needed, and terminate them when they aren't.

[AWS Instance Scheduler](#) allows you to configure the stop and start of your Amazon EC2 and Amazon RDS instances at defined times so that you can meet the demand for the same resources within a consistent time pattern such as every day user access Amazon EC2 instances at eight in the morning that they don't need after six at night. This solution helps reduce operational cost by stopping resources that are not in use and starting them when they are needed.



Cost optimization with AWS Instance Scheduler.

You can also easily configure schedules for your Amazon EC2 instances across your accounts and Regions with a simple user interface (UI) using AWS Systems Manager Quick Setup. You can schedule Amazon EC2 or Amazon RDS instances with AWS Instance Scheduler and you can stop and start existing instances. However, you cannot stop and start instances which are part of your Auto Scaling group (ASG) or that manage services such as Amazon Redshift or Amazon OpenSearch Service. Auto Scaling groups have their own scheduling for the instances in the group and these instances are created.

[AWS Auto Scaling](#) helps you adjust your capacity to maintain steady, predictable performance at the lowest possible cost to meet changing demand. It is a fully managed and free service to scale the capacity of your application that integrates with Amazon EC2 instances and Spot Fleets, Amazon ECS, Amazon DynamoDB, and Amazon Aurora. Auto Scaling provides automatic resource discovery to help find resources in your workload that can be configured, it has built-in scaling strategies to optimize performance, costs, or a balance between the two, and provides predictive scaling to assist with regularly occurring spikes.

There are multiple scaling options available to scale your Auto Scaling group:

- Maintain current instance levels at all times
- Scale manually
- Scale based on a schedule
- Scale based on demand
- Use predictive scaling

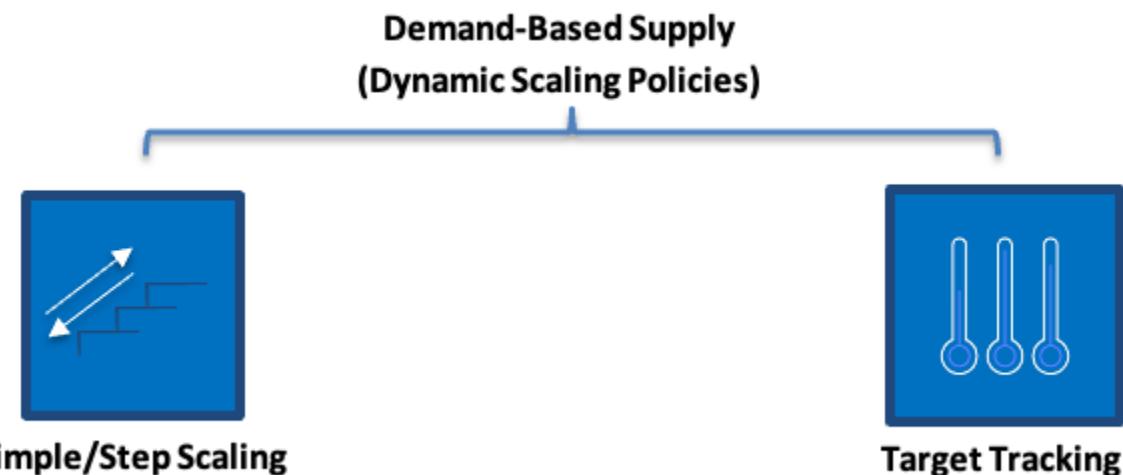
Auto Scaling policies differ and can be categorized as dynamic and scheduled scaling policies. Dynamic policies are manual or dynamic scaling which, scheduled or predictive scaling. You can use scaling policies for dynamic, scheduled, and predictive scaling. You can also use metrics and alarms from [Amazon CloudWatch](#) to trigger scaling events for your workload. We recommend you use [launch templates](#), which allow you to access the latest features and improvements. Not all Auto Scaling features are available when you use launch configurations. For example, you cannot create an Auto Scaling group that launches both Spot and On-Demand Instances or that specifies multiple instance types. You must use a launch template to configure these features. When using launch templates, we recommend you version each one. With versioning of launch templates, you can create a subset of the full set of parameters. Then, you can reuse it to create other versions of the same launch template.

You can use AWS Auto Scaling or incorporate scaling in your code with [AWS APIs or SDKs](#). This reduces your overall workload costs by removing the operational cost from manually making changes to your environment, and changes can be performed much faster. This also matches your workload resourcing to your demand at any time. In order to follow this best practice and supply resources dynamically for your organization, you should understand horizontal and vertical scaling in the AWS Cloud, as well as the nature of the applications running on Amazon EC2 instances. It is better for your Cloud Financial Management team to work with technical teams to follow this best practice.

[Elastic Load Balancing \(Elastic Load Balancing\)](#) helps you scale by distributing demand across multiple resources. With using ASG and Elastic Load Balancing, you can manage incoming requests by optimally routing traffic so that no one instance is overwhelmed in an Auto Scaling group. The requests would be distributed among all the targets of a target group in a round-robin fashion without consideration for capacity or utilization.

Typical metrics can be standard Amazon EC2 metrics, such as CPU utilization, network throughput, and Elastic Load Balancing observed request and response latency. When possible, you should use a metric that is indicative of customer experience, typically a custom metric that might originate from application code within your workload. To elaborate how to meet the demand dynamically in this document, we will group Auto Scaling into two categories as demand-based and time-based supply models and deep dive into each.

Demand-based supply: Take advantage of elasticity of the cloud to supply resources to meet changing demand by relying on near real-time demand state. For demand-based supply, use APIs or service features to programmatically vary the amount of cloud resources in your architecture. This allows you to scale components in your architecture and increase the number of resources during demand spikes to maintain performance and decrease capacity when demand subsides to reduce costs.

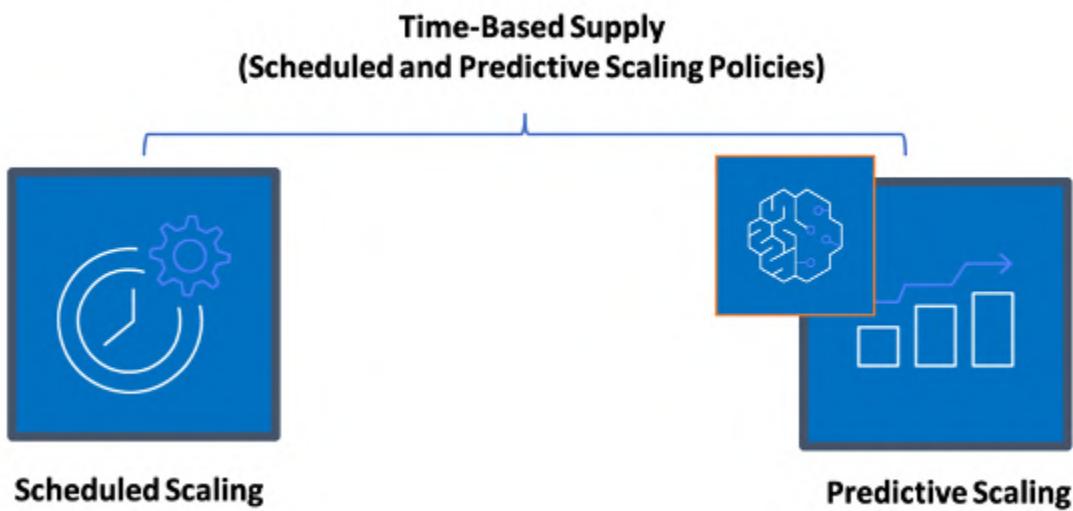


Demand-based dynamic scaling policies

- **Simple/Step Scaling:** Monitors metrics and adds/removes instances as per steps defined by the customers manually.
- **Target Tracking:** Thermostat-like control mechanism that automatically adds or removes instances to maintain metrics at a customer defined target.

When architecting with a demand-based approach keep in mind two key considerations. First, understand how quickly you must provision new resources. Second, understand that the size of margin between supply and demand will shift. You must be ready to cope with the rate of change in demand and also be ready for resource failures.

Time-based supply: A time-based approach aligns resource capacity to demand that is predictable or well-defined by time. This approach is typically not dependent upon utilization levels of the resources. A time-based approach ensures that resources are available at the specific time they are required and can be provided without any delays due to start-up procedures and system or consistency checks. Using a time-based approach, you can provide additional resources or increase capacity during busy periods.



Time-based scaling policies

You can use scheduled or predictive auto scaling to implement a time-based approach. Workloads can be scheduled to scale out or in at defined times (for example, the start of business hours), making resources available when users arrive or demand increases. Predictive scaling uses patterns to scale out while scheduled scaling uses pre-defined times to scale out. You can also use [attribute-based instance type selection \(ABS\) strategy](#) in Auto Scaling groups, which lets you express your instance requirements as a set of attributes, such as vCPU, memory, and storage. This also allows you to automatically use newer generation instance types when they are released and access a broader range of capacity with Amazon EC2 Spot Instances. Amazon EC2 Fleet and Amazon EC2 Auto Scaling select and launch instances that fit the specified attributes, removing the need to manually pick instance types.

You can also leverage the [AWS APIs and SDKs](#) and [AWS CloudFormation](#) to automatically provision and decommission entire environments as you need them. This approach is well suited for development or test environments that run only in defined business hours or periods of time. You can use APIs to scale the size of resources within an environment (vertical scaling). For example, you could scale up a production workload by changing the instance size or class. This can be achieved by stopping and starting the instance and selecting the different instance size or class. This technique can also be applied to other resources, such as Amazon EBS Elastic Volumes, which can be modified to increase size, adjust performance (IOPS) or change the volume type while in use.

When architecting with a time-based approach keep in mind two key considerations. First, how consistent is the usage pattern? Second, what is the impact if the pattern changes? You can increase the accuracy of predictions by monitoring your workloads and by using business intelligence. If you see significant changes in the usage pattern, you can adjust the times to ensure that coverage is provided.

Implementation steps

- **Configure scheduled scaling:** For predictable changes in demand, time-based scaling can provide the correct number of resources in a timely manner. It is also useful if resource creation and configuration is not fast enough to respond to changes on demand. Using the workload analysis configure scheduled scaling using AWS Auto Scaling. To configure time-based scheduling, you can use predictive scaling or scheduled scaling to increase the number of Amazon EC2 instances in your Auto Scaling groups in advance according to expected or predictable load changes.
- **Configure predictive scaling:** Predictive scaling allows you to increase the number of Amazon EC2 instances in your Auto Scaling group in advance of daily and weekly patterns in traffic flows. If you have regular traffic spikes and applications that take a long time to start, you should consider using predictive scaling. Predictive scaling can help you scale faster by initializing capacity before projected load compared to dynamic scaling alone, which is reactive in nature. For example, if users start using your workload with the start of the business hours and don't use after hours, then predictive scaling can add capacity before the business hours which eliminates delay of dynamic scaling to react to changing traffic.
- **Configure dynamic automatic scaling:** To configure scaling based on active workload metrics, use Auto Scaling. Use the analysis and configure Auto Scaling to launch on the correct resource levels, and verify that the workload scales in the required time. You can launch and automatically scale a fleet of On-Demand Instances and Spot Instances within a single Auto Scaling group. In addition to receiving discounts for using Spot Instances, you can use Reserved Instances or a Savings Plan to receive discounted rates of the regular On-Demand Instance pricing. All of these factors combined help you to optimize your cost savings for Amazon EC2 instances and help you get the desired scale and performance for your application.

Resources

Related documents:

- [AWS Auto Scaling](#)

- [AWS Instance Scheduler](#)
- Scale the size of your Auto Scaling group
- [Getting Started with Amazon EC2 Auto Scaling](#)
- [Getting started with Amazon SQS](#)
- [Scheduled Scaling for Amazon EC2 Auto Scaling](#)
- [Predictive scaling for Amazon EC2 Auto Scaling](#)

Related videos:

- [Target Tracking Scaling Policies for Auto Scaling](#)
- [AWS Instance Scheduler](#)

Related examples:

- [Attribute based Instance Type Selection for Auto Scaling for Amazon EC2 Fleet](#)
- [Optimizing Amazon Elastic Container Service for cost using scheduled scaling](#)
- [Predictive Scaling with Amazon EC2 Auto Scaling](#)
- [How do I use Instance Scheduler with AWS CloudFormation to schedule Amazon EC2 instances?](#)

Optimize over time

Questions

- [COST 10. How do you evaluate new services?](#)
- [COST 11. How do you evaluate the cost of effort?](#)

COST 10. How do you evaluate new services?

As AWS releases new services and features, it's a best practice to review your existing architectural decisions to verify they continue to be the most cost effective.

Best practices

- [COST10-BP01 Develop a workload review process](#)
- [COST10-BP02 Review and analyze this workload regularly](#)

COST10-BP01 Develop a workload review process

Develop a process that defines the criteria and process for workload review. The review effort should reflect potential benefit. For example, core workloads or workloads with a value of over ten percent of the bill are reviewed quarterly or every six months, while workloads below ten percent are reviewed annually.

Level of risk exposed if this best practice is not established: High

Implementation guidance

To have the most cost-efficient workload, you must regularly review the workload to know if there are opportunities to implement new services, features, and components. To achieve overall lower costs the process must be proportional to the potential amount of savings. For example, workloads that are 50% of your overall spend should be reviewed more regularly, and more thoroughly, than workloads that are five percent of your overall spend. Factor in any external factors or volatility. If the workload services a specific geography or market segment, and change in that area is predicted, more frequent reviews could lead to cost savings. Another factor in review is the effort to implement changes. If there are significant costs in testing and validating changes, reviews should be less frequent.

Factor in the long-term cost of maintaining outdated and legacy, components and resources and the inability to implement new features into them. The current cost of testing and validation may exceed the proposed benefit. However, over time, the cost of making the change may significantly increase as the gap between the workload and the current technologies increases, resulting in even larger costs. For example, the cost of moving to a new programming language may not currently be cost effective. However, in five years time, the cost of people skilled in that language may increase, and due to workload growth, you would be moving an even larger system to the new language, requiring even more effort than previously.

Break down your workload into components, assign the cost of the component (an estimate is sufficient), and then list the factors (for example, effort and external markets) next to each component. Use these indicators to determine a review frequency for each workload. For example, you may have webservers as a high cost, low change effort, and high external factors, resulting in high frequency of review. A central database may be medium cost, high change effort, and low external factors, resulting in a medium frequency of review.

Define a process to evaluate new services, design patterns, resource types, and configurations to optimize your workload cost as they become available. Similar to [performance pillar review](#) and

[reliability pillar review](#) processes, identify, validate, and prioritize optimization and improvement activities and issue remediation and incorporate this into your backlog.

Implementation steps

- **Define review frequency:** Define how frequently the workload and its components should be reviewed. Allocate time and resources to continual improvement and review frequency to improve the efficiency and optimization of your workload. This is a combination of factors and may differ from workload to workload within your organization and between components in the workload. Common factors include the importance to the organization measured in terms of revenue or brand, the total cost of running the workload (including operation and resource costs), the complexity of the workload, how easy is it to implement a change, any software licensing agreements, and if a change would incur significant increases in licensing costs due to punitive licensing. Components can be defined functionally or technically, such as web servers and databases, or compute and storage resources. Balance the factors accordingly and develop a period for the workload and its components. You may decide to review the full workload every 18 months, review the web servers every six months, the database every 12 months, compute and short-term storage every six months, and long-term storage every 12 months.
- **Define review thoroughness:** Define how much effort is spent on the review of the workload or workload components. Similar to the review frequency, this is a balance of multiple factors. Evaluate and prioritize opportunities for improvement to focus efforts where they provide the greatest benefits while estimating how much effort is required for these activities. If the expected outcomes do not satisfy the goals, and required effort costs more, then iterate using alternative courses of action. Your review processes should include dedicated time and resources to make continuous incremental improvements possible. As an example, you may decide to spend one week of analysis on the database component, one week of analysis for compute resources, and four hours for storage reviews.

Resources

Related documents:

- [AWS News Blog](#)
- [Types of Cloud Computing](#)
- [What's New with AWS](#)

Related examples:

- [AWS Support Proactive Services](#)
- [Regular workload reviews for SAP workloads](#)

COST10-BP02 Review and analyze this workload regularly

Existing workloads are regularly reviewed based on each defined process to find out if new services can be adopted, existing services can be replaced, or workloads can be re-architected.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

AWS is constantly adding new features so you can experiment and innovate faster with the latest technology. [AWS What's New](#) details how AWS is doing this and provides a quick overview of AWS services, features, and Regional expansion announcements as they are released. You can dive deeper into the launches that have been announced and use them for your review and analyze of your existing workloads. To realize the benefits of new AWS services and features, you review on your workloads and implement new services and features as required. This means you may need to replace existing services you use for your workload, or modernize your workload to adopt these new AWS services. For example, you might review your workloads and replace the messaging component with Amazon Simple Email Service. This removes the cost of operating and maintaining a fleet of instances, while providing all the functionality at a reduced cost.

To analyze your workload and highlight potential opportunities, you should consider not only new services but also new ways of building solutions. Review the [This is My Architecture](#) videos on AWS to learn about other customers' architecture designs, their challenges and their solutions. Check the [All-In series](#) to find out real world applications of AWS services and customer stories. You can also watch the [Back to Basics](#) video series that explains, examines, and breaks down basic cloud architecture pattern best practices. Another source is [How to Build This](#) videos, which are designed to assist people with big ideas on how to bring their minimum viable product (MVP) to life using AWS services. It is a way for builders from all over the world who have a strong idea to gain architectural guidance from experienced AWS Solutions Architects. Finally, you can review the [Getting Started](#) resource materials, which has step by step tutorials.

Before starting your review process, follow your business' requirements for the workload, security and data privacy requirements in order to use specific service or Region and performance requirements while following your agreed review process.

Implementation steps

- **Regularly review the workload:** Using your defined process, perform reviews with the frequency specified. Verify that you spend the correct amount of effort on each component. This process would be similar to the initial design process where you selected services for cost optimization. Analyze the services and the benefits they would bring, this time factor in the cost of making the change, not just the long-term benefits.
- **Implement new services:** If the outcome of the analysis is to implement changes, first perform a baseline of the workload to know the current cost for each output. Implement the changes, then perform an analysis to confirm the new cost for each output.

Resources

Related documents:

- [AWS News Blog](#)
- [What's New with AWS](#)
- [AWS Documentation](#)
- [AWS Getting Started](#)
- [AWS General Resources](#)

Related videos:

- [AWS - This is My Architecture](#)
- [AWS - Back to Basics](#)
- [AWS - All-In series](#)
- [How to Build This](#)

COST 11. How do you evaluate the cost of effort?

Best practices

- [COST11-BP01 Perform automation for operations](#)

COST11-BP01 Perform automation for operations

Evaluate cost of effort for operations on cloud. Quantify reduction in time and effort for admin tasks, deployment and other operations using automation. Evaluate the required time and cost for the effort of operations and automate admin tasks to reduce the human effort where possible.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Automating operations improves consistency and scalability, provides more visibility, reliability, and flexibility, reduces costs, and accelerates innovation by freeing up human resources and improving metrics. It reduces the frequency of manual tasks, improves efficiency, and benefits enterprises by delivering a consistent and reliable experience when deploying, administering, or operating workloads. You can free up infrastructure resources from manual operational tasks and use them for higher value tasks and innovations, thereby improving business outcomes. Enterprises require a proven, tested way to manage their workloads in the cloud. That solution must be secure, fast, and cost effective, with minimum risk and maximum reliability.

Start by prioritizing your operations based on required effort by looking at overall operations cost in the cloud. For example, how long does it take to deploy new resources in the cloud, make optimization changes to existing ones, or implement necessary configurations? Look at the total cost of human actions by factoring in cost of operations and management. Prioritize automations for admin tasks to reduce the human effort. Review effort should reflect the potential benefit. For example, time spent performing tasks manually as opposed to automatically. Prioritize automating repetitive, high value activities. Activities that pose a higher risk of human error are typically the better place to start automating as the risk often poses an unwanted additional operational cost (like operations team working extra hours).

Using AWS services, tools, or third-party products, you can choose which AWS automations to implement and customize for your specific requirements. The following table shows some of the core operation functions and capabilities you can achieve with AWS services to automate administration and operation:

- [AWS Audit Manager](#): Continually audit your AWS usage to simplify risk and compliance assessment
- [AWS Backup](#): Centrally manage and automate data protection.
- [AWS Config](#): Configure compute resources, assess, audit, and evaluate configurations and resource inventory.

- [AWS CloudFormation](#): Launch highly available resources with infrastructure as code.
- [AWS CloudTrail](#): IT change management, compliance, and control.
- [Amazon EventBridge](#): Schedule events and launch AWS Lambda to take action.
- [AWS Lambda](#): Automate repetitive processes by initiating them with events or by running them on a fixed schedule with Amazon EventBridge.
- [AWS Systems Manager](#): Start and stop workloads, patch operating systems, automate configuration, and ongoing management.
- [AWS Step Functions](#): Schedule jobs and automate workflows.
- [AWS Service Catalog](#): Template consumption and infrastructure as code with compliance and control.

Consider the time savings that will allow your team to focus on retiring technical debt, innovation, and value-adding features. For example, you might need to lift and shift your on-premises environment into the cloud as rapidly as possible and optimize later. It is worth exploring the savings you could realize by using fully managed services by AWS that remove or reduce license costs such as [Amazon Relational Database Service](#), [Amazon EMR](#), [Amazon WorkSpaces](#), and [Amazon SageMaker](#). Managed services remove the operational and administrative burden of maintaining a service, which allows you to focus on innovation. Additionally, because managed services operate at cloud scale, they can offer a lower cost per transaction or service.

If you would like to adopt automations immediately with using AWS products and service and if don't have skills in your organization, reach out to [AWS Managed Services \(AMS\)](#), [AWS Professional Services](#), or [AWS Partners](#) to increase adoption of automation and improve your operational excellence in the cloud.

[AWS Managed Services \(AMS\)](#) is a service that operates AWS infrastructure on behalf of enterprise customers and partners. It provides a secure and compliant environment that you can deploy your workloads onto. AMS uses enterprise cloud operating models with automation to allow you to meet your organization requirements, move into the cloud faster, and reduce your on-going management costs.

[AWS Professional Services](#) can also help you achieve your desired business outcomes and automate operations with AWS. AWS Professional Services provides global specialty practices to support your efforts in focused areas of enterprise cloud computing. Specialty practices deliver targeted guidance through best practices, frameworks, tools, and services across solution, technology, and

industry subject areas. They help customers to deploy automated, robust, agile IT operations, and governance capabilities optimized for the cloud center.

Implementation steps

- **Build once and deploy many:** Use infrastructure-as-code such as AWS CloudFormation, AWS SDK, or AWS Command Line Interface (AWS CLI) to deploy once and use many times for same environment or for disaster recovery scenarios. Tag while deploying to track your consumption as defined in other best practices. Use [AWS Launch Wizard](#) to reduce the time to deploy many popular enterprise workloads. AWS Launch Wizard guides you through the sizing, configuration, and deployment of enterprise workloads following AWS best practices. You can also use the [AWS Service Catalog](#), which helps you create and manage infrastructure-as-code approved templates for use on AWS so anyone can discover approved, self-service cloud resources.
- **Automate operations:** Run routine operations automatically without human intervention. Using AWS services and tools, you can choose which AWS automations to implement and customize for your specific requirements. For example, use [EC2 Image Builder](#) for building, testing, and deployment of virtual machine and container images for use on AWS or on-premises. If your desired action cannot be done with AWS services or you need more complex actions with filtering resources, then automate your operations with using [AWS CLI](#) or AWS SDK tools. AWS CLI provides the ability to automate the entire process of controlling and managing AWS services via scripts without using the AWS Console. Select your preferred AWS SDKs to interact with AWS services. For other code examples, see [AWS SDK Code examples repository](#).

Resources

Related documents:

- [Modernizing operations in the AWS Cloud](#)
- [AWS Services for Automation](#)
- [AWS Systems Manager Automation](#)
- [AWS automations for SAP administration and operations](#)
- [AWS Managed Services](#)
- [AWS Professional Services](#)
- [Infrastructure and automation](#)

Related examples:

- [Reinventing automated operations \(Part I\)](#)
- [Reinventing automated operations \(Part II\)](#)
- [AWS automations for SAP administration and operations](#)
- [IT Automations with AWS Lambda](#)
- [AWS Code Examples Repository](#)
- [AWS Samples](#)

Sustainability

The Sustainability pillar includes understanding the impacts of the services used, quantifying impacts through the entire workload lifecycle, and applying design principles and best practices to reduce these impacts when building cloud workloads. You can find prescriptive guidance on implementation in the [Sustainability Pillar whitepaper](#).

Best practice areas

- [Region selection](#)
- [Alignment to demand](#)
- [Software and architecture](#)
- [Data](#)
- [Hardware and services](#)
- [Process and culture](#)

Region selection

Question

- [SUS 1 How do you select Regions for your workload?](#)

SUS 1 How do you select Regions for your workload?

The choice of Region for your workload significantly affects its KPIs, including performance, cost, and carbon footprint. To effectively improve these KPIs, you should choose Regions for your workloads based on both business requirements and sustainability goals.

Best practices

- [SUS01-BP01 Choose Region based on both business requirements and sustainability goals](#)

SUS01-BP01 Choose Region based on both business requirements and sustainability goals

Choose a Region for your workload based on both your business requirements and sustainability goals to optimize its KPIs, including performance, cost, and carbon footprint.

Common anti-patterns:

- You select the workload's Region based on your own location.
- You consolidate all workload resources into one geographic location.

Benefits of establishing this best practice: Placing a workload close to Amazon renewable energy projects or Regions with low published carbon intensity can help to lower the carbon footprint of a cloud workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

The AWS Cloud is a constantly expanding network of Regions and points of presence (PoP), with a global network infrastructure linking them together. The choice of Region for your workload significantly affects its KPIs, including performance, cost, and carbon footprint. To effectively improve these KPIs, you should choose Regions for your workload based on both your business requirements and sustainability goals.

Implementation steps

- Follow these steps to assess and shortlist potential Regions for your workload based on your business requirements including compliance, available features, cost, and latency:
 - Confirm that these Regions are compliant, based on your required local regulations.
 - Use the [AWS Regional Services Lists](#) to check if the Regions have the services and features you need to run your workload.
 - Calculate the cost of the workload on each Region using the [AWS Pricing Calculator](#).
 - Test the network latency between your end user locations and each AWS Region.
- Choose Regions near Amazon renewable energy projects and Regions where the grid has a published carbon intensity that is lower than other locations (or Regions).

- Identify your relevant sustainability guidelines to track and compare year-to-year carbon emissions based on [Greenhouse Gas Protocol](#) (market-based and location based methods).
- Choose region based on method you use to track carbon emissions. For more detail on choosing a Region based on your sustainability guidelines, see [How to select a Region for your workload based on sustainability goals](#).

Resources

Related documents:

- [Understanding your carbon emission estimations](#)
- [Amazon Around the Globe](#)
- [Renewable Energy Methodology](#)
- [What to Consider when Selecting a Region for your Workloads](#)

Related videos:

- [Architecting sustainably and reducing your AWS carbon footprint](#)

Alignment to demand

Question

- [SUS 2 How do you align cloud resources to your demand?](#)

SUS 2 How do you align cloud resources to your demand?

The way users and applications consume your workloads and other resources can help you identify improvements to meet sustainability goals. Scale infrastructure to continually match demand and verify that you use only the minimum resources required to support your users. Align service levels to customer needs. Position resources to limit the network required for users and applications to consume them. Remove unused assets. Provide your team members with devices that support their needs and minimize their sustainability impact.

Best practices

- [SUS02-BP01 Scale workload infrastructure dynamically](#)

- [SUS02-BP02 Align SLAs with sustainability goals](#)
- [SUS02-BP03 Stop the creation and maintenance of unused assets](#)
- [SUS02-BP04 Optimize geographic placement of workloads based on their networking requirements](#)
- [SUS02-BP05 Optimize team member resources for activities performed](#)
- [SUS02-BP06 Implement buffering or throttling to flatten the demand curve](#)

SUS02-BP01 Scale workload infrastructure dynamically

Use elasticity of the cloud and scale your infrastructure dynamically to match supply of cloud resources to demand and avoid overprovisioned capacity in your workload.

Common anti-patterns:

- You do not scale your infrastructure with user load.
- You manually scale your infrastructure all the time.
- You leave increased capacity after a scaling event instead of scaling back down.

Benefits of establishing this best practice: Configuring and testing workload elasticity help to efficiently match supply of cloud resources to demand and avoid overprovisioned capacity. You can take advantage of elasticity in the cloud to automatically scale capacity during and after demand spikes to make sure you are only using the right number of resources needed to meet your business requirements.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

The cloud provides the flexibility to expand or reduce your resources dynamically through a variety of mechanisms to meet changes in demand. Optimally matching supply to demand delivers the lowest environmental impact for a workload.

Demand can be fixed or variable, requiring metrics and automation to make sure that management does not become burdensome. Applications can scale vertically (up or down) by modifying the instance size, horizontally (in or out) by modifying the number of instances, or a combination of both.

You can use a number of different approaches to match supply of resources with demand.

- **Target-tracking approach:** Monitor your scaling metric and automatically increase or decrease capacity as you need it.
- **Predictive scaling:** Scale in anticipation of daily and weekly trends.
- **Schedule-based approach:** Set your own scaling schedule according to predictable load changes.
- **Service scaling:** Pick services (like serverless) that are natively scaling by design or provide auto scaling as a feature.

Identify periods of low or no utilization and scale resources to remove excess capacity and improve efficiency.

Implementation steps

- Elasticity matches the supply of resources you have against the demand for those resources. Instances, containers, and functions provide mechanisms for elasticity, either in combination with automatic scaling or as a feature of the service. AWS provides a range of auto scaling mechanisms to ensure that workloads can scale down quickly and easily during periods of low user load. Here are some examples of auto scaling mechanisms:

Auto scaling mechanism	Where to use
Amazon EC2 Auto Scaling	Use to verify you have the correct number of Amazon EC2 instances available to handle the user load for your application.
Application Auto Scaling	Use to automatically scale the resources for individual AWS services beyond Amazon EC2, such as Lambda functions or Amazon Elastic Container Service (Amazon ECS) services.
Kubernetes Cluster Autoscaler	Use to automatically scale Kubernetes clusters on AWS.

- Scaling is often discussed related to compute services like Amazon EC2 instances or AWS Lambda functions. Consider the configuration of non-compute services like [Amazon DynamoDB](#) read and write capacity units or [Amazon Kinesis Data Streams](#) shards to match the demand.
- Verify that the metrics for scaling up or down are validated against the type of workload being deployed. If you are deploying a video transcoding application, 100% CPU utilization is expected

and should not be your primary metric. You can use a [customized metric](#) (such as memory utilization) for your scaling policy if required. To choose the right metrics, consider the following guidance for Amazon EC2:

- The metric should be a valid utilization metric and describe how busy an instance is.
- The metric value must increase or decrease proportionally to the number of instances in the Auto Scaling group.
- Use [dynamic scaling](#) instead of [manual scaling](#) for your Auto Scaling group. We also recommend that you use [target tracking scaling policies](#) in your dynamic scaling.
- Verify that workload deployments can handle both scale-out and scale-in events. Create test scenarios for scale-in events to verify that the workload behaves as expected and does not affect the user experience (like losing sticky sessions). You can use [Activity history](#) to verify a scaling activity for an Auto Scaling group.
- Evaluate your workload for predictable patterns and proactively scale as you anticipate predicted and planned changes in demand. With predictive scaling, you can eliminate the need to overprovision capacity. For more detail, see [Predictive Scaling with Amazon EC2 Auto Scaling](#).

Resources

Related documents:

- [Getting Started with Amazon EC2 Auto Scaling](#)
- [Predictive Scaling for EC2, Powered by Machine Learning](#)
- [Analyze user behavior using Amazon OpenSearch Service, Amazon Data Firehose and Kibana](#)
- [What is Amazon CloudWatch?](#)
- [Monitoring DB load with Performance Insights on Amazon RDS](#)
- [Introducing Native Support for Predictive Scaling with Amazon EC2 Auto Scaling](#)
- [Introducing Karpenter - An Open-Source, High-Performance Kubernetes Cluster Autoscaler](#)
- [Deep Dive on Amazon ECS Cluster Auto Scaling](#)

Related videos:

- [Build a cost-, energy-, and resource-efficient compute environment](#)
- [Better, faster, cheaper compute: Cost-optimizing Amazon EC2 \(CMP202-R1\)](#)

Related examples:

- [Lab: Amazon EC2 Auto Scaling Group Examples](#)
- [Lab: Implement Autoscaling with Karpenter](#)

SUS02-BP02 Align SLAs with sustainability goals

Review and optimize workload service-level agreements (SLA) based on your sustainability goals to minimize the resources required to support your workload while continuing to meet business needs.

Common anti-patterns:

- Workload SLAs are unknown or ambiguous.
- You define your SLA just for availability and performance.
- You use the same design pattern (like Multi-AZ architecture) for all your workloads.

Benefits of establishing this best practice: Aligning SLAs with sustainability goals leads to optimal resource usage while meeting business needs.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

SLAs define the level of service expected from a cloud workload, such as response time, availability, and data retention. They influence the architecture, resource usage, and environmental impact of a cloud workload. At a regular cadence, review SLAs and make trade-offs that significantly reduce resource usage in exchange for acceptable decreases in service levels.

Implementation steps

- Define or redesign SLAs that support your sustainability goals while meeting your business requirements, not exceeding them.
- Make trade-offs that significantly reduce sustainability impacts in exchange for acceptable decreases in service levels.
 - **Sustainability and reliability:** Highly available workloads tend to consume more resources.
 - **Sustainability and performance:** Using more resources to boost performance could have a higher environmental impact.

- **Sustainability and security:** Overly secure workloads could have a higher environmental impact.
- Use design patterns such as [microservices on AWS](#) that prioritize business-critical functions and allow lower service levels (such as response time or recovery time objectives) for non-critical functions.

Resources

Related documents:

- [AWS Service Level Agreements \(SLAs\)](#)
- [Importance of Service Level Agreement for SaaS Providers](#)

Related videos:

- [Delivering sustainable, high-performing architectures](#)
- [Build a cost-, energy-, and resource-efficient compute environment](#)

SUS02-BP03 Stop the creation and maintenance of unused assets

Decommission unused assets in your workload to reduce the number of cloud resources required to support your demand and minimize waste.

Common anti-patterns:

- You do not analyze your application for assets that are redundant or no longer required.
- You do not remove assets that are redundant or no longer required.

Benefits of establishing this best practice: Removing unused assets frees resources and improves the overall efficiency of the workload.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Unused assets consume cloud resources like storage space and compute power. By identifying and eliminating these assets, you can free up these resources, resulting in a more efficient cloud

architecture. Perform regular analysis on application assets such as pre-compiled reports, datasets, static images, and asset access patterns to identify redundancy, underutilization, and potential decommission targets. Remove those redundant assets to reduce the resource waste in your workload.

Implementation steps

- Use monitoring tools to identify static assets that are no longer required.
- Before removing any asset, evaluate the impact of removing it on the architecture.
- Develop a plan and remove assets that are no longer required.
- Consolidate overlapping generated assets to remove redundant processing.
- Update your applications to no longer produce and store assets that are not required.
- Instruct third parties to stop producing and storing assets managed on your behalf that are no longer required.
- Instruct third parties to consolidate redundant assets produced on your behalf.
- Regularly review your workload to identify and remove unused assets.

Resources

Related documents:

- [Optimizing your AWS Infrastructure for Sustainability, Part II: Storage](#)
- [How do I terminate active resources that I no longer need on my AWS account?](#)

SUS02-BP04 Optimize geographic placement of workloads based on their networking requirements

Select cloud location and services for your workload that reduce the distance network traffic must travel and decrease the total network resources required to support your workload.

Common anti-patterns:

- You select the workload's Region based on your own location.
- You consolidate all workload resources into one geographic location.
- All traffic flows through your existing data centers.

Benefits of establishing this best practice: Placing a workload close to its users provides the lowest latency while decreasing data movement across the network and reducing environmental impact.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

The AWS Cloud infrastructure is built around location options such as Regions, Availability Zones, placement groups, and edge locations such as [AWS Outposts](#) and [AWS Local Zones](#). These location options are responsible for maintaining connectivity between application components, cloud services, edge networks and on-premises data centers.

Analyze the network access patterns in your workload to identify how to use these cloud location options and reduce the distance network traffic must travel.

Implementation steps

- Analyze network access patterns in your workload to identify how users use your application.
 - Use monitoring tools, such as [Amazon CloudWatch](#) and [AWS CloudTrail](#), to gather data on network activities.
 - Analyze the data to identify the network access pattern.
- Select the Regions for your workload deployment based on the following key elements:
 - **Your Sustainability goal:** as explained in [Region selection](#).
 - **Where your data is located:** For data-heavy applications (such as big data and machine learning), application code should run as close to the data as possible.
 - **Where your users are located:** For user-facing applications, choose a Region (or Regions) close to your workload's users.
 - **Other constraints:** Consider constraints such as cost and compliance as explained in [What to Consider when Selecting a Region for your Workloads](#).
- Use local caching or [AWS Caching Solutions](#) for frequently used assets to improve performance, reduce data movement, and lower environmental impact.

Service	When to use
Amazon CloudFront	Use to cache static content such as images, scripts, and videos, as well as dynamic

Service	When to use
	content such as API responses or web applications.
Amazon ElastiCache	Use to cache content for web applications.
DynamoDB Accelerator	Use to add in-memory acceleration to your DynamoDB tables.

- Use services that can help you run code closer to users of your workload:

Service	When to use
Lambda@Edge	Use for compute-heavy operations that are initiated when objects are not in the cache.
Amazon CloudFront Functions	Use for simple use cases like HTTP(s) request or response manipulations that can be initiated by short-lived functions.
AWS IoT Greengrass	Use to run local compute, messaging, and data caching for connected devices.

- Use connection pooling to allow for connection reuse and reduce required resources.
- Use distributed data stores that don't rely on persistent connections and synchronous updates for consistency to serve regional populations.
- Replace pre-provisioned static network capacity with shared dynamic capacity, and share the sustainability impact of network capacity with other subscribers.

Resources

Related documents:

- [Optimizing your AWS Infrastructure for Sustainability, Part III: Networking](#)
- [Amazon ElastiCache Documentation](#)
- [What is Amazon CloudFront?](#)
- [Amazon CloudFront Key Features](#)

Related videos:

- [Demystifying data transfer on AWS](#)
- [Scaling network performance on next-gen Amazon EC2 instances](#)

Related examples:

- [AWS Networking Workshops](#)
- [Architecting for sustainability - Minimize data movement across networks](#)

SUS02-BP05 Optimize team member resources for activities performed

Optimize resources provided to team members to minimize the environmental sustainability impact while supporting their needs.

Common anti-patterns:

- You ignore the impact of devices used by your team members on the overall efficiency of your cloud application.
- You manually manage and update resources used by team members.

Benefits of establishing this best practice: Optimizing team member resources improves the overall efficiency of cloud-enabled applications.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Understand the resources your team members use to consume your services, their expected lifecycle, and the financial and sustainability impact. Implement strategies to optimize these resources. For example, perform complex operations, such as rendering and compilation, on highly utilized scalable infrastructure instead of on underutilized high-powered single-user systems.

Implementation steps

- Provision workstations and other devices to align with how they're used.
- Use virtual desktops and application streaming to limit upgrade and device requirements.
- Move processor or memory-intensive tasks to the cloud to use its elasticity.

- Evaluate the impact of processes and systems on your device lifecycle, and select solutions that minimize the requirement for device replacement while satisfying business requirements.
- Implement remote management for devices to reduce required business travel.
 - [AWS Systems Manager Fleet Manager](#) is a unified user interface (UI) experience that helps you remotely manage your nodes running on AWS or on premises.

Resources

Related documents:

- [What is Amazon WorkSpaces?](#)
- [Cost Optimizer for Amazon WorkSpaces](#)
- [Amazon AppStream 2.0 Documentation](#)
- [NICE DCV](#)

Related videos:

- [Managing cost for Amazon WorkSpaces on AWS](#)

SUS02-BP06 Implement buffering or throttling to flatten the demand curve

Buffering and throttling flatten the demand curve and reduce the provisioned capacity required for your workload.

Common anti-patterns:

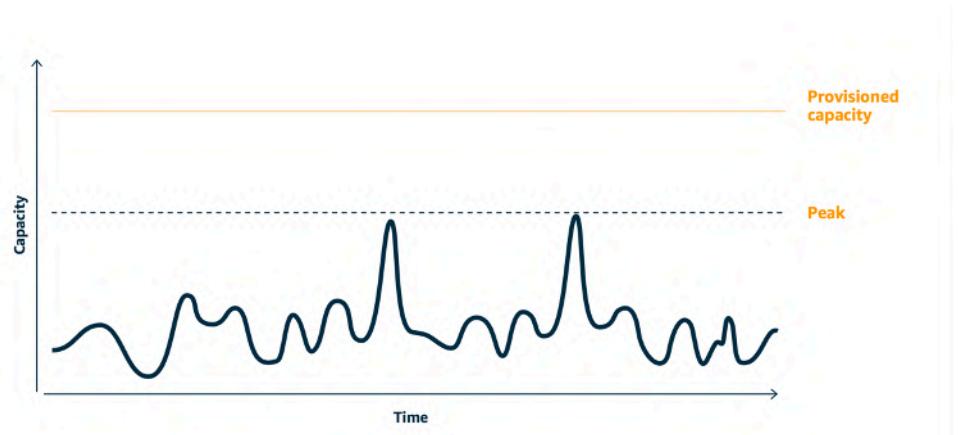
- You process the client requests immediately while it is not needed.
- You do not analyze the requirements for client requests.

Benefits of establishing this best practice: Flattening the demand curve reduce the required provisioned capacity for the workload. Reducing the provisioned capacity means less energy consumption and less environmental impact.

Level of risk exposed if this best practice is not established: Low

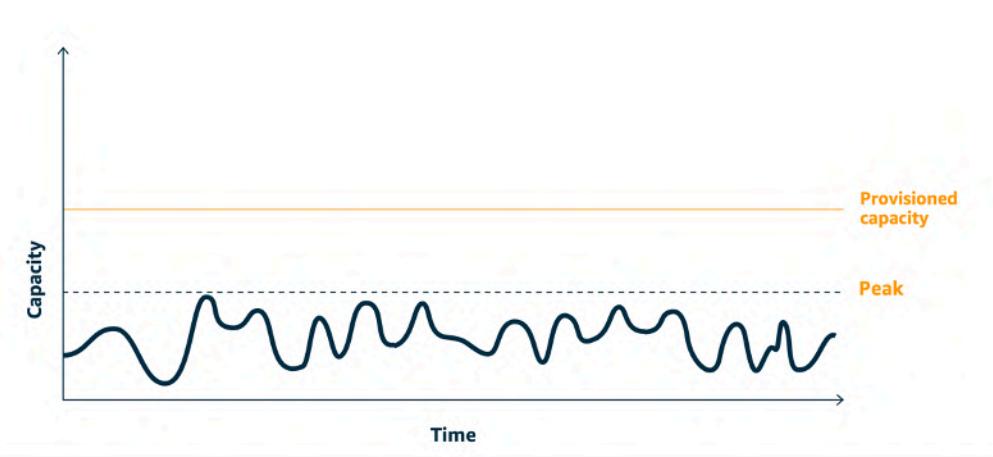
Implementation guidance

Flattening the workload demand curve can help you to reduce the provisioned capacity for a workload and reduce its environmental impact. Assume a workload with the demand curve shown in below figure. This workload has two peaks, and to handle those peaks, the resource capacity as shown by orange line is provisioned. The resources and energy used for this workload is not indicated by the area under the demand curve, but the area under the provisioned capacity line, as provisioned capacity is needed to handle those two peaks.



Demand curve with two distinct peaks that require high provisioned capacity.

You can use buffering or throttling to modify the demand curve and smooth out the peaks, which means less provisioned capacity and less energy consumed. Implement throttling when your clients can perform retries. Implement buffering to store the request and defer processing until a later time.



Throttling's effect on the demand curve and provisioned capacity.

Implementation steps

- Analyze the client requests to determine how to respond to them. Questions to consider include:
 - Can this request be processed asynchronously?
 - Does the client have retry capability?
- If the client has retry capability, then you can implement throttling, which tells the source that if it cannot service the request at the current time, it should try again later.
 - You can use [Amazon API Gateway](#) to implement throttling.
- For clients that cannot perform retries, a buffer needs to be implemented to flatten the demand curve. A buffer defers request processing, allowing applications that run at different rates to communicate effectively. A buffer-based approach uses a queue or a stream to accept messages from producers. Messages are read by consumers and processed, allowing the messages to run at the rate that meets the consumers' business requirements.
 - [Amazon Simple Queue Service \(Amazon SQS\)](#) is a managed service that provides queues that allow a single consumer to read individual messages.
 - [Amazon Kinesis](#) provides a stream that allows many consumers to read the same messages.
- Analyze the overall demand, rate of change, and required response time to right size the throttle or buffer required.

Resources

Related documents:

- [Getting started with Amazon SQS](#)
- [Application integration Using Queues and Messages](#)

Related videos:

- [Choosing the Right Messaging Service for Your Distributed App](#)

Software and architecture

Question

- [SUS 3 How do you take advantage of software and architecture patterns to support your sustainability goals?](#)

SUS 3 How do you take advantage of software and architecture patterns to support your sustainability goals?

Implement patterns for performing load smoothing and maintaining consistent high utilization of deployed resources to minimize the resources consumed. Components might become idle from lack of use because of changes in user behavior over time. Revise patterns and architecture to consolidate under-utilized components to increase overall utilization. Retire components that are no longer required. Understand the performance of your workload components, and optimize the components that consume the most resources. Be aware of the devices that your customers use to access your services, and implement patterns to minimize the need for device upgrades.

Best practices

- [SUS03-BP01 Optimize software and architecture for asynchronous and scheduled jobs](#)
- [SUS03-BP02 Remove or refactor workload components with low or no use](#)
- [SUS03-BP03 Optimize areas of code that consume the most time or resources](#)
- [SUS03-BP04 Optimize impact on devices and equipment](#)
- [SUS03-BP05 Use software patterns and architectures that best support data access and storage patterns](#)

SUS03-BP01 Optimize software and architecture for asynchronous and scheduled jobs

Use efficient software and architecture patterns such as queue-driven to maintain consistent high utilization of deployed resources.

Common anti-patterns:

- You overprovision the resources in your cloud workload to meet unforeseen spikes in demand.
- Your architecture does not decouple senders and receivers of asynchronous messages by a messaging component.

Benefits of establishing this best practice:

- Efficient software and architecture patterns minimize the unused resources in your workload and improve the overall efficiency.
- You can scale the processing independently of the receiving of asynchronous messages.

- Through a messaging component, you have relaxed availability requirements that you can meet with fewer resources.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Use efficient architecture patterns such as [event-driven architecture](#) that result in even utilization of components and minimize overprovisioning in your workload. Using efficient architecture patterns minimizes idle resources from lack of use due to changes in demand over time.

Understand the requirements of your workload components and adopt architecture patterns that increase overall utilization of resources. Retire components that are no longer required.

Implementation steps

- Analyze the demand for your workload to determine how to respond to those.
- For requests or jobs that don't require synchronous responses, use queue-driven architectures and auto scaling workers to maximize utilization. Here are some examples of when you might consider queue-driven architecture:

Queuing mechanism	Description
AWS Batch job queues	AWS Batch jobs are submitted to a job queue where they reside until they can be scheduled to run in a compute environment.
Amazon Simple Queue Service and Amazon EC2 Spot Instances	Pairing Amazon SQS and Spot Instances to build fault tolerant and efficient architecture.

- For requests or jobs that can be processed anytime, use scheduling mechanisms to process jobs in batch for more efficiency. Here are some examples of scheduling mechanisms on AWS:

Scheduling mechanism	Description
Amazon EventBridge Scheduler	A capability from Amazon EventBridge that allows you to create, run, and manage scheduled tasks at scale.

Scheduling mechanism	Description
AWS Glue time-based schedule	Define a time-based schedule for your crawlers and jobs in AWS Glue.
Amazon Elastic Container Service (Amazon ECS) scheduled tasks	Amazon ECS supports creating scheduled tasks. Scheduled tasks use Amazon EventBridge rules to run tasks either on a schedule or in a response to an EventBridge event.
Instance Scheduler	Configure start and stop schedules for your Amazon EC2 and Amazon Relational Database Service instances.

- If you use polling and webhooks mechanisms in your architecture, replace those with events. Use [event-driven architectures](#) to build highly efficient workloads.
- Leverage [serverless on AWS](#) to eliminate over-provisioned infrastructure.
- Right size individual components of your architecture to prevent idling resources waiting for input.

Resources

Related documents:

- [What is Amazon Simple Queue Service?](#)
- [What is Amazon MQ?](#)
- [Scaling based on Amazon SQS](#)
- [What is AWS Step Functions?](#)
- [What is AWS Lambda?](#)
- [Using AWS Lambda with Amazon SQS](#)
- [What is Amazon EventBridge?](#)

Related videos:

- [Moving to event-driven architectures](#)

SUS03-BP02 Remove or refactor workload components with low or no use

Remove components that are unused and no longer required, and refactor components with little utilization to minimize waste in your workload.

Common anti-patterns:

- You do not regularly check the utilization level of individual components of your workload.
- You do not check and analyze recommendations from AWS rightsizing tools such as [AWS Compute Optimizer](#).

Benefits of establishing this best practice: Removing unused components minimizes waste and improves the overall efficiency of your cloud workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Review your workload to identify idle or unused components. This is an iterative improvement process which can be initiated by changes in demand or the release of a new cloud service. For example, a significant drop in [AWS Lambda](#) function run time can be an indicator of a need to lower the memory size. Also, as AWS releases new services and features, the optimal services and architecture for your workload may change.

Continually monitor workload activity and look for opportunities to improve the utilization level of individual components. By removing idle components and performing rightsizing activities, you meet your business requirements with the fewest cloud resources.

Implementation steps

- Monitor and capture the utilization metrics for critical components of your workload (like CPU utilization, memory utilization, or network throughput in [Amazon CloudWatch metrics](#)).
- For stable workloads, check AWS rightsizing tools such as [AWS Compute Optimizer](#) at regular intervals to identify idle, unused, or underutilized components.
- For ephemeral workloads, evaluate utilization metrics to identify idle, unused, or underutilized components.
- Retire components and associated assets (like Amazon ECR images) that are no longer needed.

- Refactor or consolidate underutilized components with other resources to improve utilization efficiency. For example, you can provision multiple small databases on a single [Amazon RDS](#) database instance instead of running databases on individual under-utilized instances.
- Understand the [resources provisioned by your workload to complete a unit of work](#).

Resources

Related documents:

- [AWS Trusted Advisor](#)
- [What is Amazon CloudWatch?](#)
- [Automated Cleanup of Unused Images in Amazon ECR](#)

Related examples:

- [Well-Architected Lab - Rightsizing with AWS Compute Optimizer](#)
- [Well-Architected Lab - Optimize Hardware Patterns and Observe Sustainability KPIs](#)

SUS03-BP03 Optimize areas of code that consume the most time or resources

Optimize your code that runs within different components of your architecture to minimize resource usage while maximizing performance.

Common anti-patterns:

- You ignore optimizing your code for resource usage.
- You usually respond to performance issues by increasing the resources.
- Your code review and development process does not track performance changes.

Benefits of establishing this best practice: Using efficient code minimizes resource usage and improves performance.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

It is crucial to examine every functional area, including the code for a cloud architected application, to optimize its resource usage and performance. Continually monitor your workload's performance

in build environments and production and identify opportunities to improve code snippets that have particularly high resource usage. Adopt a regular review process to identify bugs or anti-patterns within your code that use resources inefficiently. Leverage simple and efficient algorithms that produce the same results for your use case.

Implementation steps

- While developing your workloads, adopt an automated code review process to improve quality and identify bugs and anti-patterns.
 - [Automate code reviews with Amazon CodeGuru Reviewer](#)
 - [Detecting concurrency bugs with Amazon CodeGuru](#)
 - [Raising code quality for Python applications using Amazon CodeGuru](#)
- As you run your workloads, monitor resources to identify components with high resource requirements per unit of work as targets for code reviews.
- For code reviews, use a code profiler to identify the areas of code that use the most time or resources as targets for optimization.
 - [Reducing your organization's carbon footprint with Amazon CodeGuru Profiler](#)
 - [Understanding memory usage in your Java application with Amazon CodeGuru Profiler](#)
 - [Improving customer experience and reducing cost with Amazon CodeGuru Profiler](#)
- Use the most efficient operating system and programming language for the workload. For details on energy efficient programming languages (including Rust), see [Sustainability with Rust](#).
- Replace computationally intensive algorithms with simpler and more efficient version that produce the same result.
- Remove unnecessary code such as sorting and formatting.

Resources

Related documents:

- [What is Amazon CodeGuru Profiler?](#)
- [FPGA instances](#)
- [The AWS SDKs on Tools to Build on AWS](#)

Related videos:

- [Improve Code Efficiency Using Amazon CodeGuru Profiler](#)
- [Automate Code Reviews and Application Performance Recommendations with Amazon CodeGuru](#)

SUS03-BP04 Optimize impact on devices and equipment

Understand the devices and equipment used in your architecture and use strategies to reduce their usage. This can minimize the overall environmental impact of your cloud workload.

Common anti-patterns:

- You ignore the environmental impact of devices used by your customers.
- You manually manage and update resources used by customers.

Benefits of establishing this best practice: Implementing software patterns and features that are optimized for customer device can reduce the overall environmental impact of cloud workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Implementing software patterns and features that are optimized for customer devices can reduce the environmental impact in several ways:

- Implementing new features that are backward compatible can reduce the number of hardware replacements.
- Optimizing an application to run efficiently on devices can help to reduce their energy consumption and extend their battery life (if they are powered by battery).
- Optimizing an application for devices can also reduce the data transfer over the network.

Understand the devices and equipment used in your architecture, their expected lifecycle, and the impact of replacing those components. Implement software patterns and features that can help to minimize the device energy consumption, the need for customers to replace the device and also upgrade it manually.

Implementation steps

- Inventory the devices used in your architecture. Devices can be mobile, tablet, IOT devices, smart light, or even smart devices in a factory.
- Optimize the application running on the devices:
 - Use strategies such as running tasks in the background to reduce their energy consumption.
 - Account for network bandwidth and latency when building payloads, and implement capabilities that help your applications work well on low bandwidth, high latency links.
 - Convert payloads and files into optimized formats required by devices. For example, you can use [Amazon Elastic Transcoder](#) or [AWS Elemental MediaConvert](#) to convert large, high quality digital media files into formats that users can play back on mobile devices, tablets, web browsers, and connected televisions.
 - Perform computationally intense activities server-side (such as image rendering), or use application streaming to improve the user experience on older devices.
 - Segment and paginate output, especially for interactive sessions, to manage payloads and limit local storage requirements.
- Use automated over-the-air (OTA) mechanism to deploy updates to one or more devices.
 - You can use a [CI/CD pipeline](#) to update mobile applications.
 - You can use [AWS IoT Device Management](#) to remotely manage connected devices at scale.
- To test new features and updates, use managed device farms with representative sets of hardware and iterate development to maximize the devices supported. For more details, see [SUS06-BP04 Use managed device farms for testing](#).

Resources

Related documents:

- [What is AWS Device Farm?](#)
- [Amazon AppStream 2.0 Documentation](#)
- [NICE DCV](#)
- [OTA tutorial for updating firmware on devices running FreeRTOS](#)

Related videos:

- [Introduction to AWS Device Farm](#)

SUS03-BP05 Use software patterns and architectures that best support data access and storage patterns

Understand how data is used within your workload, consumed by your users, transferred, and stored. Use software patterns and architectures that best support data access and storage to minimize the compute, networking, and storage resources required to support the workload.

Common anti-patterns:

- You assume that all workloads have similar data storage and access patterns.
- You only use one tier of storage, assuming all workloads fit within that tier.
- You assume that data access patterns will stay consistent over time.
- Your architecture supports a potential high data access burst, which results in the resources remaining idle most of the time.

Benefits of establishing this best practice: Selecting and optimizing your architecture based on data access and storage patterns will help decrease development complexity and increase overall utilization. Understanding when to use global tables, data partitioning, and caching will help you decrease operational overhead and scale based on your workload needs.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Use software and architecture patterns that aligns best to your data characteristics and access patterns. For example, use [modern data architecture on AWS](#) that allows you to use purpose-built services optimized for your unique analytics use cases. These architecture patterns allow for efficient data processing and reduce the resource usage.

Implementation steps

- Analyze your data characteristics and access patterns to identify the correct configuration for your cloud resources. Key characteristics to consider include:
 - **Data type:** structured, semi-structured, unstructured
 - **Data growth:** bounded, unbounded
 - **Data durability:** persistent, ephemeral, transient
 - **Access patterns** reads or writes, update frequency, spiky, or consistent

- Use architecture patterns that best support data access and storage patterns.
 - [Let's Architect! Modern data architectures](#)
 - [Databases on AWS: The Right Tool for the Right Job](#)
- Use technologies that work natively with compressed data.
- Use purpose-built [analytics services](#) for data processing in your architecture.
- Use the database engine that best supports your dominant query pattern. Manage your database indexes to ensure efficient querying. For further details, see [AWS Databases](#).
- Select network protocols that reduce the amount of network capacity consumed in your architecture.

Resources

Related documents:

- [Athena Compression Support file formats](#)
- [COPY from columnar data formats with Amazon Redshift](#)
- [Converting Your Input Record Format in Firehose](#)
- [Format Options for ETL Inputs and Outputs in AWS Glue](#)
- [Improve query performance on Amazon Athena by Converting to Columnar Formats](#)
- [Loading compressed data files from Amazon S3 with Amazon Redshift](#)
- [Monitoring DB load with Performance Insights on Amazon Aurora](#)
- [Monitoring DB load with Performance Insights on Amazon RDS](#)
- [Amazon S3 Intelligent-Tiering storage class](#)

Related videos:

- [Building modern data architectures on AWS](#)

Data

Question

- [SUS 4 How do you take advantage of data management policies and patterns to support your sustainability goals?](#)

SUS 4 How do you take advantage of data management policies and patterns to support your sustainability goals?

Implement data management practices to reduce the provisioned storage required to support your workload, and the resources required to use it. Understand your data, and use storage technologies and configurations that more effectively support the business value of the data and how it's used. Lifecycle data to more efficient, less performant storage when requirements decrease, and delete data that's no longer required.

Best practices

- [SUS04-BP01 Implement a data classification policy](#)
- [SUS04-BP02 Use technologies that support data access and storage patterns](#)
- [SUS04-BP03 Use policies to manage the lifecycle of your datasets](#)
- [SUS04-BP04 Use elasticity and automation to expand block storage or file system](#)
- [SUS04-BP05 Remove unneeded or redundant data](#)
- [SUS04-BP06 Use shared file systems or storage to access common data](#)
- [SUS04-BP07 Minimize data movement across networks](#)
- [SUS04-BP08 Back up data only when difficult to recreate](#)

SUS04-BP01 Implement a data classification policy

Classify data to understand its criticality to business outcomes and choose the right energy-efficient storage tier to store the data.

Common anti-patterns:

- You do not identify data assets with similar characteristics (such as sensitivity, business criticality, or regulatory requirements) that are being processed or stored.
- You have not implemented a data catalog to inventory your data assets.

Benefits of establishing this best practice: Implementing a data classification policy allows you to determine the most energy-efficient storage tier for data.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Data classification involves identifying the types of data that are being processed and stored in an information system owned or operated by an organization. It also involves making a determination on the criticality of the data and the likely impact of a data compromise, loss, or misuse.

Implement data classification policy by working backwards from the contextual use of the data and creating a categorization scheme that takes into account the level of criticality of a given dataset to an organization's operations.

Implementation steps

- Conduct an inventory of the various data types that exist for your workload.
 - For more detail on data classification categories, see [Data Classification whitepaper](#).
- Determine criticality, confidentiality, integrity, and availability of data based on risk to the organization. Use these requirements to group data into one of the data classification tiers that you adopt.
 - As an example, see [Four simple steps to classify your data and secure your startup](#).
- Periodically audit your environment for untagged and unclassified data, and classify and tag the data appropriately.
 - As an example, see [Data Catalog and crawlers in AWS Glue](#).
 - Establish a data catalog that provides audit and governance capabilities.
 - Determine and document the handling procedures for each data class.
 - Use automation to continually audit your environment to identify untagged and unclassified data, and classify and tag the data appropriately.

Resources

Related documents:

- [Leveraging AWS Cloud to Support Data Classification](#)
- [Tag policies from AWS Organizations](#)

Related videos:

- [Enabling agility with data governance on AWS](#)

SUS04-BP02 Use technologies that support data access and storage patterns

Use storage technologies that best support how your data is accessed and stored to minimize the resources provisioned while supporting your workload.

Common anti-patterns:

- You assume that all workloads have similar data storage and access patterns.
- You only use one tier of storage, assuming all workloads fit within that tier.
- You assume that data access patterns will stay consistent over time.

Benefits of establishing this best practice: Selecting and optimizing your storage technologies based on data access and storage patterns will help you reduce the required cloud resources to meet your business needs and improve the overall efficiency of cloud workload.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Select the storage solution that aligns best to your access patterns, or consider changing your access patterns to align with the storage solution to maximize performance efficiency.

Implementation steps

- Evaluate your data characteristics and access pattern to collect the key characteristics of your storage needs. Key characteristics to consider include:
 - **Data type:** structured, semi-structured, unstructured
 - **Data growth:** bounded, unbounded
 - **Data durability:** persistent, ephemeral, transient
 - **Access patterns:** reads or writes, frequency, spiky, or consistent
- Migrate data to the appropriate storage technology that supports your data characteristics and access pattern. Here are some examples of AWS storage technologies and their key characteristics:

Type	Technology	Key characteristics
Object storage	Amazon S3	An object storage service with unlimited scalabili

Type	Technology	Key characteristics
		ty, high availability, and multiple options for accessibility. Transferring and accessing objects in and out of Amazon S3 can use a service, such as Transfer Acceleration or Access Points , to support your location, security needs, and access patterns.
Archiving storage	Amazon S3 Glacier	Storage class of Amazon S3 built for data-archiving.
Shared file system	Amazon Elastic File System (Amazon EFS)	Mountable file system that can be accessed by multiple types of compute solutions . Amazon EFS automatically grows and shrinks storage and is performance-optimized to deliver consistent low latencies.
Shared file system	Amazon FSx	Built on the latest AWS compute solutions to support four commonly used file systems: NetApp ONTAP, OpenZFS, Windows File Server, and Lustre. Amazon FSx latency , throughput , and IOPS vary per file system and should be considered when selecting the right file system for your workload needs.

Type	Technology	Key characteristics
Block storage	Amazon Elastic Block Store (Amazon EBS)	Scalable, high-performance block-storage service designed for Amazon Elastic Compute Cloud (Amazon EC2). Amazon EBS includes SSD-backed storage for transactional, IOPS-intensive workloads and HDD-backed storage for throughput-intensive workloads.
Relational database	Amazon Aurora , Amazon RDS , Amazon Redshift	Designed to support ACID (atomicity, consistency, isolation, durability) transactions and maintain referential integrity and strong data consistency. Many traditional applications, enterprise resource planning (ERP), customer relationship management (CRM), and ecommerce systems use relational databases to store their data.
Key-value database	Amazon DynamoDB	Optimized for common access patterns, typically to store and retrieve large volumes of data. High-traffic web apps, ecommerce systems, and gaming applications are typical use-cases for key-value databases.

- For storage systems that are a fixed size, such as Amazon EBS or Amazon FSx, monitor the available storage space and automate storage allocation on reaching a threshold. You can leverage Amazon CloudWatch to collect and analyze different metrics for [Amazon EBS](#) and [Amazon FSx](#).
- Amazon S3 Storage Classes can be configured at the object level and a single bucket can contain objects stored across all of the storage classes.
- You can also use Amazon S3 Lifecycle policies to automatically transition objects between storage classes or remove data without any application changes. In general, you have to make a trade-off between resource efficiency, access latency, and reliability when considering these storage mechanisms.

Resources

Related documents:

- [Amazon EBS volume types](#)
- [Amazon EC2 instance store](#)
- [Amazon S3 Intelligent-Tiering](#)
- [Amazon EBS I/O Characteristics](#)
- [Using Amazon S3 storage classes](#)
- [What is Amazon S3 Glacier?](#)

Related videos:

- [Architectural Patterns for Data Lakes on AWS](#)
- [Deep dive on Amazon EBS \(STG303-R1\)](#)
- [Optimize your storage performance with Amazon S3 \(STG343\)](#)
- [Building modern data architectures on AWS](#)

Related examples:

- [Amazon EFS CSI Driver](#)
- [Amazon EBS CSI Driver](#)
- [Amazon EFS Utilities](#)

- [Amazon EBS Autoscale](#)
- [Amazon S3 Examples](#)

SUS04-BP03 Use policies to manage the lifecycle of your datasets

Manage the lifecycle of all of your data and automatically enforce deletion to minimize the total storage required for your workload.

Common anti-patterns:

- You manually delete data.
- You do not delete any of your workload data.
- You do not move data to more energy-efficient storage tiers based on its retention and access requirements.

Benefits of establishing this best practice: Using data lifecycle policies ensures efficient data access and retention in a workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Datasets usually have different retention and access requirements during their lifecycle. For example, your application may need frequent access to some datasets for a limited period of time. After that, those datasets are infrequently accessed.

To efficiently manage your datasets throughout their lifecycle, configure lifecycle policies, which are rules that define how to handle datasets.

With Lifecycle configuration rules, you can tell the specific storage service to transition a dataset to more energy-efficient storage tiers, archive it, or delete it.

Implementation steps

- [Classify datasets in your workload.](#)
- Define handling procedures for each data class.
- Set automated lifecycle policies to enforce lifecycle rules. Here are some examples of how to set up automated lifecycle policies for different AWS storage services:

Storage service	How to set automated lifecycle policies
Amazon S3	<p>You can use Amazon S3 Lifecycle to manage your objects throughout their lifecycle. If your access patterns are unknown, changing, or unpredictable, you can use Amazon S3 Intelligent-Tiering, which monitors access patterns and automatically moves objects that have not been accessed to lower-cost access tiers. You can leverage Amazon S3 Storage Lens metrics to identify optimization opportunities and gaps in lifecycle management.</p>
Amazon Elastic Block Store	<p>You can use Amazon Data Lifecycle Manager to automate the creation, retention, and deletion of Amazon EBS snapshots and Amazon EBS-backed AMIs.</p>
Amazon Elastic File System	<p>Amazon EFS lifecycle management automatically manages file storage for your file systems.</p>
Amazon Elastic Container Registry	<p>Amazon ECR lifecycle policies automate the cleanup of your container images by expiring images based on age or count.</p>
AWS Elemental MediaStore	<p>You can use an object lifecycle policy that governs how long objects should be stored in the MediaStore container.</p>

- Delete unused volumes, snapshots, and data that is out of its retention period. Leverage native service features like Amazon DynamoDB Time To Live or Amazon CloudWatch log retention for deletion.
- Aggregate and compress data where applicable based on lifecycle rules.

Resources

Related documents:

- [Optimize your Amazon S3 Lifecycle rules with Amazon S3 Storage Class Analysis](#)
- [Evaluating Resources with AWS Config Rules](#)

Related videos:

- [Simplify Your Data Lifecycle and Optimize Storage Costs With Amazon S3 Lifecycle](#)
- [Reduce Your Storage Costs Using Amazon S3 Storage Lens](#)

SUS04-BP04 Use elasticity and automation to expand block storage or file system

Use elasticity and automation to expand block storage or file system as data grows to minimize the total provisioned storage.

Common anti-patterns:

- You procure large block storage or file system for future need.
- You overprovision the input and output operations per second (IOPS) of your file system.
- You do not monitor the utilization of your data volumes.

Benefits of establishing this best practice: Minimizing over-provisioning for storage system reduces the idle resources and improves the overall efficiency of your workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Create block storage and file systems with size allocation, throughput, and latency that are appropriate for your workload. Use elasticity and automation to expand block storage or file system as data grows without having to over-provision these storage services.

Implementation steps

- For fixed size storage like [Amazon EBS](#), verify that you are monitoring the amount of storage used versus the overall storage size and create automation, if possible, to increase the storage size when reaching a threshold.

- Use elastic volumes and managed block data services to automate allocation of additional storage as your persistent data grows. As an example, you can use [Amazon EBS Elastic Volumes](#) to change volume size, volume type, or adjust the performance of your Amazon EBS volumes.
- Choose the right storage class, performance mode, and throughput mode for your file system to address your business need, not exceeding that.
 - [Amazon EFS performance](#)
 - [Amazon EBS volume performance on Linux instances](#)
- Set target levels of utilization for your data volumes, and resize volumes outside of expected ranges.
- Right size read-only volumes to fit the data.
- Migrate data to object stores to avoid provisioning the excess capacity from fixed volume sizes on block storage.
- Regularly review elastic volumes and file systems to terminate idle volumes and shrink over-provisioned resources to fit the current data size.

Resources

Related documents:

- [Amazon FSx Documentation](#)
- [What is Amazon Elastic File System?](#)

Related videos:

- [Deep Dive on Amazon EBS Elastic Volumes](#)
- [Amazon EBS and Snapshot Optimization Strategies for Better Performance and Cost Savings](#)
- [Optimizing Amazon EFS for cost and performance, using best practices](#)

SUS04-BP05 Remove unneeded or redundant data

Remove unneeded or redundant data to minimize the storage resources required to store your datasets.

Common anti-patterns:

- You duplicate data that can be easily obtained or recreated.

- You back up all data without considering its criticality.
- You only delete data irregularly, on operational events, or not at all.
- You store data redundantly irrespective of the storage service's durability.
- You turn on Amazon S3 versioning without any business justification.

Benefits of establishing this best practice: Removing unneeded data reduces the storage size required for your workload and the workload environmental impact.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Do not store data that you do not need. Automate the deletion of unneeded data. Use technologies that deduplicate data at the file and block level. Leverage native data replication and redundancy features of services.

Implementation steps

- Evaluate if you can avoid storing data by using existing publicly available datasets in [AWS Data Exchange](#) and [Open Data on AWS](#).
- Use mechanisms that can deduplicate data at the block and object level. Here are some examples of how to deduplicate data on AWS:

Storage service	Deduplication mechanism
Amazon S3	Use AWS Lake Formation FindMatches to find matching records across a dataset (including ones without identifiers) by using the new FindMatches ML Transform.
Amazon FSx	Use data deduplication on Amazon FSx for Windows.
Amazon Elastic Block Store snapshots	Snapshots are incremental backups, which means that only the blocks on the device that have changed after your most recent snapshot are saved.

- Analyze the data access to identify unneeded data. Automate lifecycle policies. Leverage native service features like [Amazon DynamoDB Time To Live](#), [Amazon S3 Lifecycle](#), or [Amazon CloudWatch log retention](#) for deletion.
- Use data virtualization capabilities on AWS to maintain data at its source and avoid data duplication.
 - [Cloud Native Data Virtualization on AWS](#)
 - [Lab: Optimize Data Pattern Using Amazon Redshift Data Sharing](#)
- Use backup technology that can make incremental backups.
- Leverage the durability of [Amazon S3](#) and [replication of Amazon EBS](#) to meet your durability goals instead of self-managed technologies (such as a redundant array of independent disks (RAID)).
- Centralize log and trace data, deduplicate identical log entries, and establish mechanisms to tune verbosity when needed.
- Pre-populate caches only where justified.
- Establish cache monitoring and automation to resize the cache accordingly.
- Remove out-of-date deployments and assets from object stores and edge caches when pushing new versions of your workload.

Resources

Related documents:

- [Change log data retention in CloudWatch Logs](#)
- [Data deduplication on Amazon FSx for Windows File Server](#)
- [Features of Amazon FSx for ONTAP including data deduplication](#)
- [Invalidating Files on Amazon CloudFront](#)
- [Using AWS Backup to back up and restore Amazon EFS file systems](#)
- [What is Amazon CloudWatch Logs?](#)
- [Working with backups on Amazon RDS](#)

Related videos:

- [Fuzzy Matching and Deduplicating Data with ML Transforms for AWS Lake Formation](#)

Related examples:

- [How do I analyze my Amazon S3 server access logs using Amazon Athena?](#)

SUS04-BP06 Use shared file systems or storage to access common data

Adopt shared file systems or storage to avoid data duplication and allow for more efficient infrastructure for your workload.

Common anti-patterns:

- You provision storage for each individual client.
- You do not detach data volume from inactive clients.
- You do not provide access to storage across platforms and systems.

Benefits of establishing this best practice: Using shared file systems or storage allows for sharing data to one or more consumers without having to copy the data. This helps to reduce the storage resources required for the workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

If you have multiple users or applications accessing the same datasets, using shared storage technology is crucial to use efficient infrastructure for your workload. Shared storage technology provides a central location to store and manage datasets and avoid data duplication. It also enforces consistency of the data across different systems. Moreover, shared storage technology allows for more efficient use of compute power, as multiple compute resources can access and process data at the same time in parallel.

Fetch data from these shared storage services only as needed and detach unused volumes to free up resources.

Implementation steps

- Migrate data to shared storage when the data has multiple consumers. Here are some examples of shared storage technology on AWS:

Storage option	When to use
Amazon EBS Multi-Attach	Amazon EBS Multi-Attach allows you to attach a single Provisioned IOPS SSD (io1 or io2) volume to multiple instances that are in the same Availability Zone.
Amazon EFS	See When to Choose Amazon EFS .
Amazon FSx	See Choosing an Amazon FSx File System .
Amazon S3	Applications that do not require a file system structure and are designed to work with object storage can use Amazon S3 as a massively scalable, durable, low-cost object storage solution.

- Copy data to or fetch data from shared file systems only as needed. As an example, you can create an [Amazon FSx for Lustre file system backed by Amazon S3](#) and only load the subset of data required for processing jobs to Amazon FSx.
- Delete data as appropriate for your usage patterns as outlined in [SUS04-BP03 Use policies to manage the lifecycle of your datasets](#).
- Detach volumes from clients that are not actively using them.

Resources

Related documents:

- [Linking your file system to an Amazon S3 bucket](#)
- [Using Amazon EFS for AWS Lambda in your serverless applications](#)
- [Amazon EFS Intelligent-Tiering Optimizes Costs for Workloads with Changing Access Patterns](#)
- [Using Amazon FSx with your on-premises data repository](#)

related videos:

- [Storage cost optimization with Amazon EFS](#)

SUS04-BP07 Minimize data movement across networks

Use shared file systems or object storage to access common data and minimize the total networking resources required to support data movement for your workload.

Common anti-patterns:

- You store all data in the same AWS Region independent of where the data users are.
- You do not optimize data size and format before moving it over the network.

Benefits of establishing this best practice: Optimizing data movement across the network reduces the total networking resources required for the workload and lowers its environmental impact.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Moving data around your organization requires compute, networking, and storage resources. Use techniques to minimize data movement and improve the overall efficiency of your workload.

Implementation steps

- Consider proximity to the data or users as a decision factor when [selecting a Region for your workload](#).
- Partition Regionally consumed services so that their Region-specific data is stored within the Region where it is consumed.
- Use efficient file formats (such as Parquet or ORC) and compress data before moving it over the network.
- Don't move unused data. Some examples that can help you avoid moving unused data:
 - Reduce API responses to only relevant data.
 - Aggregate data where detailed (record-level information is not required).
 - See [Well-Architected Lab - Optimize Data Pattern Using Amazon Redshift Data Sharing](#).
 - Consider [Cross-account data sharing in AWS Lake Formation](#).
- Use services that can help you run code closer to users of your workload.

Service	When to use
Lambda@Edge	Use for compute-heavy operations that are run when objects are not in the cache.
CloudFront Functions	Use for simple use cases such as HTTP(s) request/response manipulations that can be initiated by short-lived functions.
AWS IoT Greengrass	Run local compute, messaging, and data caching for connected devices.

Resources

Related documents:

- [Optimizing your AWS Infrastructure for Sustainability, Part III: Networking](#)
- [AWS Global Infrastructure](#)
- [Amazon CloudFront Key Features including the CloudFront Global Edge Network](#)
- [Compressing HTTP requests in Amazon OpenSearch Service](#)
- [Intermediate data compression with Amazon EMR](#)
- [Loading compressed data files from Amazon S3 into Amazon Redshift](#)
- [Serving compressed files with Amazon CloudFront](#)

Related videos:

- [Demystifying data transfer on AWS](#)

Related examples:

- [Architecting for sustainability - Minimize data movement across networks](#)

SUS04-BP08 Back up data only when difficult to recreate

Avoid backing up data that has no business value to minimize storage resources requirements for your workload.

Common anti-patterns:

- You do not have a backup strategy for your data.
- You back up data that can be easily recreated.

Benefits of establishing this best practice: Avoiding back-up of non-critical data reduces the required storage resources for the workload and lowers its environmental impact.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Avoiding the back up of unnecessary data can help lower cost and reduce the storage resources used by the workload. Only back up data that has business value or is needed to satisfy compliance requirements. Examine backup policies and exclude ephemeral storage that doesn't provide value in a recovery scenario.

Implementation steps

- Implement data classification policy as outlined in [SUS04-BP01 Implement a data classification policy](#).
- Use the criticality of your data classification and design backup strategy based on your [recovery time objective \(RTO\)](#) and [recovery point objective \(RPO\)](#). Avoid backing up non-critical data.
 - Exclude data that can be easily recreated.
 - Exclude ephemeral data from your backups.
 - Exclude local copies of data, unless the time required to restore that data from a common location exceeds your service-level agreements (SLAs).
- Use an automated solution or managed service to back up business-critical data.
 - [AWS Backup](#) is a fully-managed service that makes it easy to centralize and automate data protection across AWS services, in the cloud, and on premises. For hands-on guidance on how to create automated backups using AWS Backup, see [Well-Architected Labs - Testing Backup and Restore of Data](#).

- [Automate backups and optimize backup costs for Amazon EFS using AWS Backup.](#)

Resources

Related best practices:

- [REL09-BP01 Identify and back up all data that needs to be backed up, or reproduce the data from sources](#)
- [REL09-BP03 Perform data backup automatically](#)
- [REL13-BP02 Use defined recovery strategies to meet the recovery objectives](#)

Related documents:

- [Using AWS Backup to back up and restore Amazon EFS file systems](#)
- [Amazon EBS snapshots](#)
- [Working with backups on Amazon Relational Database Service](#)
- [APN Partner: partners that can help with backup](#)
- [AWS Marketplace: products that can be used for backup](#)
- [Backing Up Amazon EFS](#)
- [Backing Up Amazon FSx for Windows File Server](#)
- [Backup and Restore for Amazon ElastiCache for Redis](#)

Related videos:

- [AWS re:Invent 2021 - Backup, disaster recovery, and ransomware protection with AWS](#)
- [AWS Backup Demo: Cross-Account and Cross-Region Backup](#)
- [AWS re:Invent 2019: Deep dive on AWS Backup, ft. Rackspace \(STG341\)](#)

Related examples:

- [Well-Architected Lab - Testing Backup and Restore of Data](#)
- [Well-Architected Lab - Backup and Restore with Fallback for Analytics Workload](#)
- [Well-Architected Lab - Disaster Recovery - Backup and Restore](#)

Hardware and services

Question

- [SUS 5 How do you select and use cloud hardware and services in your architecture to support your sustainability goals?](#)

SUS 5 How do you select and use cloud hardware and services in your architecture to support your sustainability goals?

Look for opportunities to reduce workload sustainability impacts by making changes to your hardware management practices. Minimize the amount of hardware needed to provision and deploy, and select the most efficient hardware and services for your individual workload.

Best practices

- [SUS05-BP01 Use the minimum amount of hardware to meet your needs](#)
- [SUS05-BP02 Use instance types with the least impact](#)
- [SUS05-BP03 Use managed services](#)
- [SUS05-BP04 Optimize your use of hardware-based compute accelerators](#)

SUS05-BP01 Use the minimum amount of hardware to meet your needs

Use the minimum amount of hardware for your workload to efficiently meet your business needs.

Common anti-patterns:

- You do not monitor resource utilization.
- You have resources with a low utilization level in your architecture.
- You do not review the utilization of static hardware to determine if it should be resized.
- You do not set hardware utilization goals for your compute infrastructure based on business KPIs.

Benefits of establishing this best practice: Rightsizing your cloud resources helps to reduce a workload's environmental impact, save money, and maintain performance benchmarks.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Optimally select the total number of hardware required for your workload to improve its overall efficiency. The AWS Cloud provides the flexibility to expand or reduce the number of resources dynamically through a variety of mechanisms, such as [AWS Auto Scaling](#), and meet changes in demand. It also provides [APIs and SDKs](#) that allow resources to be modified with minimal effort. Use these capabilities to make frequent changes to your workload implementations. Additionally, use rightsizing guidelines from AWS tools to efficiently operate your cloud resource and meet your business needs.

Implementation steps

- Choose the instances type to best fit your needs.
 - [How do I choose the appropriate Amazon EC2 instance type for my workload?](#)
 - [Attribute-based instance type selection for Amazon EC2 Fleet.](#)
 - [Create an Auto Scaling group using attribute-based instance type selection.](#)
- Scale using small increments for variable workloads.
- Use multiple compute purchase options in order to balance instance flexibility, scalability, and cost savings.
 - [On-Demand Instances](#) are best suited for new, stateful, and spiky workloads which can't be instance type, location, or time flexible.
 - [Spot Instances](#) are a great way to supplement the other options for applications that are fault tolerant and flexible.
 - Leverage [Compute Savings Plans](#) for steady state workloads that allow flexibility if your needs (like AZ, Region, instance families, or instance types) change.
- Use instance and availability zone diversity to maximize application availability and take advantage of excess capacity when possible.
- Use the rightsizing recommendations from AWS tools to make adjustments on your workload.
 - [AWS Compute Optimizer](#)
 - [AWS Trusted Advisor](#)
- Negotiate service-level agreements (SLAs) that allow for a temporary reduction in capacity while automation deploys replacement resources.

Resources

Related documents:

- [Optimizing your AWS Infrastructure for Sustainability, Part I: Compute](#)
- [Attribute based Instance Type Selection for Auto Scaling for Amazon EC2 Fleet](#)
- [AWS Compute Optimizer Documentation](#)
- [Operating Lambda: Performance optimization](#)
- [Auto Scaling Documentation](#)

Related videos:

- [Build a cost-, energy-, and resource-efficient compute environment](#)

Related examples:

- [Well-Architected Lab - Rightsizing with AWS Compute Optimizer and Memory Utilization Enabled \(Level 200\)](#)

SUS05-BP02 Use instance types with the least impact

Continually monitor and use new instance types to take advantage of energy efficiency improvements.

Common anti-patterns:

- You are only using one family of instances.
- You are only using x86 instances.
- You specify one instance type in your Amazon EC2 Auto Scaling configuration.
- You use AWS instances in a manner that they were not designed for (for example, you use compute-optimized instances for a memory-intensive workload).
- You do not evaluate new instance types regularly.
- You do not check recommendations from AWS rightsizing tools such as [AWS Compute Optimizer](#).

Benefits of establishing this best practice: By using energy-efficient and right-sized instances, you are able to greatly reduce the environmental impact and cost of your workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Using efficient instances in cloud workload is crucial for lower resource usage and cost-effectiveness. Continually monitor the release of new instance types and take advantage of energy efficiency improvements, including those instance types designed to support specific workloads such as machine learning training and inference, and video transcoding.

Implementation steps

- Learn and explore instance types which can lower your workload environmental impact.
 - Subscribe to [What's New with AWS](#) to be up-to-date with the latest AWS technologies and instances.
 - Learn about different AWS instance types.
 - Learn about AWS Graviton-based instances which offer the best performance per watt of energy use in Amazon EC2 by watching [re:Invent 2020 - Deep dive on AWS Graviton2 processor-powered Amazon EC2 instances](#) and [Deep dive into AWS Graviton3 and Amazon EC2 C7g instances](#).
- Plan and transition your workload to instance types with the least impact.
 - Define a process to evaluate new features or instances for your workload. Take advantage of agility in the cloud to quickly test how new instance types can improve your workload environmental sustainability. Use proxy metrics to measure how many resources it takes you to complete a unit of work.
 - If possible, modify your workload to work with different numbers of vCPUs and different amounts of memory to maximize your choice of instance type.
 - Consider transitioning your workload to Graviton-based instances to improve the performance efficiency of your workload.
 - [AWS Graviton Fast Start](#)
 - [Considerations when transitioning workloads to AWS Graviton-based Amazon Elastic Compute Cloud instances](#)
 - [AWS Graviton2 for ISVs](#)
 - Consider selecting the AWS Graviton option in your usage of [AWS managed services](#).
 - Migrate your workload to Regions that offer instances with the least sustainability impact and still meet your business requirements.

- For machine learning workloads, take advantage of purpose-built hardware that is specific to your workload such as [AWS Trainium](#), [AWS Inferentia](#), and [Amazon EC2 DL1](#). AWS Inferentia instances such as Inf2 instances offer up to 50% better performance per watt over comparable Amazon EC2 instances.
- Use [Amazon SageMaker Inference Recommender](#) to right size ML inference endpoint.
- For spiky workloads (workloads with infrequent requirements for additional capacity), use [burstable performance instances](#).
- For stateless and fault-tolerant workloads, use [Amazon EC2 Spot Instances](#) to increase overall utilization of the cloud, and reduce the sustainability impact of unused resources.
- Operate and optimize your workload instance.
 - For ephemeral workloads, evaluate [instance Amazon CloudWatch metrics](#) such as CPUUtilization to identify if the instance is idle or under-utilized.
 - For stable workloads, check AWS rightsizing tools such as [AWS Compute Optimizer](#) at regular intervals to identify opportunities to optimize and right-size the instances.
 - [Well-Architected Lab - Rightsizing Recommendations](#)
 - [Well-Architected Lab - Rightsizing with Compute Optimizer](#)
 - [Well-Architected Lab - Optimize Hardware Patterns and Observice Sustainability KPIs](#)

Resources

Related documents:

- [Optimizing your AWS Infrastructure for Sustainability, Part I: Compute](#)
- [AWS Graviton](#)
- [Amazon EC2 DL1](#)
- [Amazon EC2 Capacity Reservation Fleets](#)
- [Amazon EC2 Spot Fleet](#)
- [Functions: Lambda Function Configuration](#)
- [Attribute-based instance type selection for Amazon EC2 Fleet](#)
- [Building Sustainable, Efficient, and Cost-Optimized Applications on AWS](#)
- [How the Contino Sustainability Dashboard Helps Customers Optimize Their Carbon Footprint](#)

Related videos:

- [Deep dive on AWS Graviton2 processor-powered Amazon EC2 instances](#)
- [Deep dive into AWS Graviton3 and Amazon EC2 C7g instances](#)
- [Build a cost-, energy-, and resource-efficient compute environment](#)

Related examples:

- [Solution: Guidance for Optimizing Deep Learning Workloads for Sustainability on AWS](#)
- [Well-Architected Lab - Rightsizing Recommendations](#)
- [Well-Architected Lab - Rightsizing with Compute Optimizer](#)
- [Well-Architected Lab - Optimize Hardware Patterns and Observe Sustainability KPIs](#)
- [Well-Architected Lab - Migrating Services to Graviton](#)

SUS05-BP03 Use managed services

Use managed services to operate more efficiently in the cloud.

Common anti-patterns:

- You use Amazon EC2 instances with low utilization to run your applications.
- Your in-house team only manages the workload, without time to focus on innovation or simplifications.
- You deploy and maintain technologies for tasks that can run more efficiently on managed services.

Benefits of establishing this best practice:

- Using managed services shifts the responsibility to AWS, which has insights across millions of customers that can help drive new innovations and efficiencies.
- Managed service distributes the environmental impact of the service across many users because of the multi-tenant control planes.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Managed services shift responsibility to AWS for maintaining high utilization and sustainability optimization of the deployed hardware. Managed services also remove the operational and administrative burden of maintaining a service, which allows your team to have more time and focus on innovation.

Review your workload to identify the components that can be replaced by AWS managed services. For example, [Amazon RDS](#), [Amazon Redshift](#), and [Amazon ElastiCache](#) provide a managed database service. [Amazon Athena](#), [Amazon EMR](#), and [Amazon OpenSearch Service](#) provide a managed analytics service.

Implementation steps

1. Inventory your workload for services and components.
2. Assess and identify components that can be replaced by managed services. Here are some examples of when you might consider using a managed service:

Task	What to use on AWS
Hosting a database	Use managed Amazon Relational Database Service (Amazon RDS) instances instead of maintaining your own Amazon RDS instances on Amazon Elastic Compute Cloud (Amazon EC2) .
Hosting a container workload	Use AWS Fargate , instead of implementing your own container infrastructure.
Hosting web apps	Use AWS Amplify Hosting as fully managed CI/CD and hosting service for static websites and server-side rendered web apps.

3. Identify dependencies and create a migrations plan. Update runbooks and playbooks accordingly.
 - The [AWS Application Discovery Service](#) automatically collects and presents detailed information about application dependencies and utilization to help you make more informed decisions as you plan your migration
4. Test the service before migrating to the managed service.

5. Use the migration plan to replace self-hosted services with managed service.
6. Continually monitor the service after the migration is complete to make adjustments as required and optimize the service.

Resources

Related documents:

- [AWS Cloud Products](#)
- [AWS Total Cost of Ownership \(TCO\) Calculator](#)
- [Amazon DocumentDB](#)
- [Amazon Elastic Kubernetes Service \(EKS\)](#)
- [Amazon Managed Streaming for Apache Kafka \(Amazon MSK\)](#)

Related videos:

- [Cloud operations at scale with AWS Managed Services](#)

SUS05-BP04 Optimize your use of hardware-based compute accelerators

Optimize your use of accelerated computing instances to reduce the physical infrastructure demands of your workload.

Common anti-patterns:

- You are not monitoring GPU usage.
- You are using a general-purpose instance for workload while a purpose-built instance can deliver higher performance, lower cost, and better performance per watt.
- You are using hardware-based compute accelerators for tasks where they're more efficient using CPU-based alternatives.

Benefits of establishing this best practice: By optimizing the use of hardware-based accelerators, you can reduce the physical-infrastructure demands of your workload.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

If you require high processing capability, you can benefit from using accelerated computing instances, which provide access to hardware-based compute accelerators such as graphics processing units (GPUs) and field programmable gate arrays (FPGAs). These hardware accelerators perform certain functions like graphic processing or data pattern matching more efficiently than CPU-based alternatives. Many accelerated workloads, such as rendering, transcoding, and machine learning, are highly variable in terms of resource usage. Only run this hardware for the time needed, and decommission them with automation when not required to minimize resources consumed.

Implementation steps

- Identify which [accelerated computing instances](#) can address your requirements.
- For machine learning workloads, take advantage of purpose-built hardware that is specific to your workload, such as [AWS Trainium](#), [AWS Inferentia](#), and [Amazon EC2 DL1](#). AWS Inferentia instances such as Inf2 instances offer up to [50% better performance per watt over comparable Amazon EC2 instances](#).
- Collect usage metric for your accelerated computing instances. For example, you can use CloudWatch agent to collect metrics such as `utilization_gpu` and `utilization_memory` for your GPUs as shown in [Collect NVIDIA GPU metrics with Amazon CloudWatch](#).
- Optimize the code, network operation, and settings of hardware accelerators to make sure that underlying hardware is fully utilized.
 - [Optimize GPU settings](#)
 - [GPU Monitoring and Optimization in the Deep Learning AMI](#)
 - [Optimizing I/O for GPU performance tuning of deep learning training in Amazon SageMaker](#)
- Use the latest high performant libraries and GPU drivers.
- Use automation to release GPU instances when not in use.

Resources

Related documents:

- [Accelerated Computing](#)
- [Let's Architect! Architecting with custom chips and accelerators](#)
- [How do I choose the appropriate Amazon EC2 instance type for my workload?](#)

- [Amazon EC2 VT1 Instances](#)
- [Choose the best AI accelerator and model compilation for computer vision inference with Amazon SageMaker](#)

Related videos:

- [How to select Amazon EC2 GPU instances for deep learning](#)
- [Deploying Cost-Effective Deep Learning Inference](#)

Process and culture

Question

- [SUS 6 How do your organizational processes support your sustainability goals?](#)

SUS 6 How do your organizational processes support your sustainability goals?

Look for opportunities to reduce your sustainability impact by making changes to your development, test, and deployment practices.

Best practices

- [SUS06-BP01 Adopt methods that can rapidly introduce sustainability improvements](#)
- [SUS06-BP02 Keep your workload up-to-date](#)
- [SUS06-BP03 Increase utilization of build environments](#)
- [SUS06-BP04 Use managed device farms for testing](#)

SUS06-BP01 Adopt methods that can rapidly introduce sustainability improvements

Adopt methods and processes to validate potential improvements, minimize testing costs, and deliver small improvements.

Common anti-patterns:

- Reviewing your application for sustainability is a task done only once at the beginning of a project.
- Your workload has become stale, as the release process is too cumbersome to introduce minor changes for resource efficiency.

- You do not have mechanisms to improve your workload for sustainability.

Benefits of establishing this best practice: By establishing a process to introduce and track sustainability improvements, you will be able to continually adopt new features and capabilities, remove issues, and improve workload efficiency.

Level of risk exposed if this best practice is not established: Medium

Implementation guidance

Test and validate potential sustainability improvements before deploying them to production. Account for the cost of testing when calculating potential future benefit of an improvement. Develop low cost testing methods to deliver small improvements.

Implementation steps

- Add requirements for sustainability improvement to your development backlog.
- Use an iterative [improvement process](#) to identify, evaluate, prioritize, test, and deploy these improvements.
- Continually improve and streamline your development processes. As an example, [Automate your software delivery process using continuous integration and delivery \(CI/CD\) pipelines](#) to test and deploy potential improvements to reduce the level of effort and limit errors caused by manual processes.
- Develop and test potential improvements using the minimum viable representative components to reduce the cost of testing.
- Continually assess the impact of improvements and make adjustment as needed.

Resources

Related documents:

- [AWS enables sustainability solutions](#)
- [Scalable agile development practices based on AWS CodeCommit](#)

Related videos:

- [Delivering sustainable, high-performing architectures](#)

Related examples:

- [Well-Architected Lab - Turning cost & usage reports into efficiency reports](#)

SUS06-BP02 Keep your workload up-to-date

Keep your workload up-to-date to adopt efficient features, remove issues, and improve the overall efficiency of your workload.

Common anti-patterns:

- You assume your current architecture is static and will not be updated over time.
- You do not have any systems or a regular cadence to evaluate if updated software and packages are compatible with your workload.

Benefits of establishing this best practice: By establishing a process to keep your workload up to date, you can adopt new features and capabilities, resolve issues, and improve workload efficiency.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Up to date operating systems, runtimes, middlewares, libraries, and applications can improve workload efficiency and make it easier to adopt more efficient technologies. Up to date software might also include features to measure the sustainability impact of your workload more accurately, as vendors deliver features to meet their own sustainability goals. Adopt a regular cadence to keep your workload up to date with the latest features and releases.

Implementation steps

- Define a process and a schedule to evaluate new features or instances for your workload. Take advantage of agility in the cloud to quickly test how new features can improve your workload to:
 - Reduce sustainability impacts.
 - Gain performance efficiencies.
 - Remove barriers for a planned improvement.
 - Improve your ability to measure and manage sustainability impacts.
- Inventory your workload software and architecture and identify components that need to be updated.

- You can use [AWS Systems Manager Inventory](#) to collect operating system (OS), application, and instance metadata from your Amazon EC2 instances and quickly understand which instances are running the software and configurations required by your software policy and which instances need to be updated.
- Understand how to update the components of your workload.

Workload component	How to update
Machine images	Use EC2 Image Builder to manage updates to Amazon Machine Images (AMIs) for Linux or Windows server images.
Container images	Use Amazon Elastic Container Registry (Amazon ECR) with your existing pipeline to manage Amazon Elastic Container Service (Amazon ECS) images .
AWS Lambda	AWS Lambda includes version management features .

- Use automation for the update process to reduce the level of effort to deploy new features and limit errors caused by manual processes.
 - You can use [CI/CD](#) to automatically update AMIs, container images, and other artifacts related to your cloud application.
 - You can use tools such as [AWS Systems Manager Patch Manager](#) to automate the process of system updates, and schedule the activity using [AWS Systems Manager Maintenance Windows](#).

Resources

Related documents:

- [AWS Architecture Center](#)
- [What's New with AWS](#)
- [AWS Developer Tools](#)

Related examples:

- [Well-Architected Labs - Inventory and Patch Management](#)
- [Lab: AWS Systems Manager](#)

SUS06-BP03 Increase utilization of build environments

Increase the utilization of resources to develop, test, and build your workloads.

Common anti-patterns:

- You manually provision or terminate your build environments.
- You keep your build environments running independent of test, build, or release activities (for example, running an environment outside of the working hours of your development team members).
- You over-provision resources for your build environments.

Benefits of establishing this best practice: By increasing the utilization of build environments, you can improve the overall efficiency of your cloud workload while allocating the resources to builders to develop, test, and build efficiently.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Use automation and infrastructure-as-code to bring build environments up when needed and take them down when not used. A common pattern is to schedule periods of availability that coincide with the working hours of your development team members. Your test environments should closely resemble the production configuration. However, look for opportunities to use instance types with burst capacity, Amazon EC2 Spot Instances, automatic scaling database services, containers, and serverless technologies to align development and test capacity with use. Limit data volume to just meet the test requirements. If using production data in test, explore possibilities of sharing data from production and not moving data across.

Implementation steps

- Use infrastructure-as-code to provision your build environments.
- Use automation to manage the lifecycle of your development and test environments and maximize the efficiency of your build resources.
- Use strategies to maximize the utilization of development and test environments.

- Use minimum viable representative environments to develop and test potential improvements.
- Use serverless technologies if possible.
- Use On-Demand Instances to supplement your developer devices.
- Use instance types with burst capacity, Spot Instances, and other technologies to align build capacity with use.
- Adopt native cloud services for secure instance shell access rather than deploying fleets of bastion hosts.
- Automatically scale your build resources depending on your build jobs.

Resources

Related documents:

- [AWS Systems Manager Session Manager](#)
- [Amazon EC2 Burstable performance instances](#)
- [What is AWS CloudFormation?](#)
- [What is AWS CodeBuild?](#)
- [Instance Scheduler on AWS](#)

Related videos:

- [Continuous Integration Best Practices](#)

SUS06-BP04 Use managed device farms for testing

Use managed device farms to efficiently test a new feature on a representative set of hardware.

Common anti-patterns:

- You manually test and deploy your application on individual physical devices.
- You do not use app testing service to test and interact with your apps (for example, Android, iOS, and web apps) on real, physical devices.

Benefits of establishing this best practice: Using managed device farms for testing cloud-enabled applications provides a number of benefits:

- They include more efficient features to test application on wide range of devices.
- They eliminate the need for in-house infrastructure for testing.
- They offer diverse device types, including older and less popular hardware, which eliminates the need for unnecessary device upgrades.

Level of risk exposed if this best practice is not established: Low

Implementation guidance

Using Managed device farms can help you to streamline the testing process for new features on a representative set of hardware. Managed device farms offer diverse device types including older, less popular hardware, and avoid customer sustainability impact from unnecessary device upgrades.

Implementation steps

- Define your testing requirements and plan (like test type, operating systems, and test schedule).
 - You can use [Amazon CloudWatch RUM](#) to collect and analyze client-side data and shape your testing plan.
- Select the managed device farm that can support your testing requirements. For example, you can use [AWS Device Farm](#) to test and understand the impact of your changes on a representative set of hardware.
- Use continuous integration/continuous deployment (CI/CD) to schedule and run your tests.
 - [Integrating AWS Device Farm with your CI/CD pipeline to run cross-browser Selenium tests](#)
 - [Building and testing iOS and iPadOS apps with AWS DevOps and mobile services](#)
- Continually review your testing results and make necessary improvements.

Resources

Related documents:

- [AWS Device Farm device list](#)
- [Viewing the CloudWatch RUM dashboard](#)

Related examples:

- [AWS Device Farm Sample App for Android](#)
- [AWS Device Farm Sample App for iOS](#)
- [Appium Web tests for AWS Device Farm](#)

Related videos:

- [Optimize applications through end user insights with Amazon CloudWatch RUM](#)

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Copyright © 2023 Amazon Web Services, Inc. or its affiliates.

AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.