AWS Prescriptive Guidance

Phased approach to modernizing applications in the AWS Cloud

AWS Prescriptive Guidance: Phased approach to modernizing applications in the AWS Cloud

Table of Contents

Phased approach to modernizing applications in the AWS Cloud

Vijay Thumma, Amazon Web Services (AWS)

December 2020 (document history (p. 12))

Modernization requires a multi-dimensional approach to adopt and consume new technology, to deliver portfolio, application, and infrastructure value faster, and to position organizations for scaling at an optimal price. It involves optimizing, maintaining applications, and operating in that modernized model without disruption, and requires that you simplify your business operations, architecture, and overall engineering practices.

Modernization is not just about applications; it requires a modern infrastructure that provides a secure and flexible operations framework. Applications and infrastructure are inseparable when it comes to business process quality, availability, and agility. Modernizing applications without taking infrastructure into account leads to high overall costs and negatively impacts performance and quality. Modern applications are built with a combination of new architecture patterns, operational models, and software delivery processes. They scale up and down from zero to millions of users, manage terabytes (if not petabytes) of data, are available globally, and respond in milliseconds. When you modernize the portfolio of workloads you manage in the Amazon Web Services (AWS) Cloud, you replatform, refactor, or replace these workloads by using containers, serverless technologies, purpose-built data stores, and software automation, to gain the fullest agility and total cost optimization (TCO) benefits offered by AWS.

This guide is for application owners, business owners, architects, technical leads, and project managers. It discusses how to develop foundational capabilities for applications selected in the modernization assessment phase, and ways to accelerate modernization efforts by using a phased approach.

The guide is part of a content series that covers the application modernization approach recommended by AWS. The series also includes:

- Strategy for modernizing applications in the AWS Cloud
- Evaluating modernization readiness for applications in the AWS Cloud
- Decomposing monoliths into microservices
- Integrating microservices by using AWS serverless services
- Enabling data persistence in microservices

Targeted business outcomes

You should expect the following outcomes from the phased approach to application modernization:

- Organizational capacity and capabilities to innovate faster, by using the build-and-prove approach and cloud-native architectures such as microservices.
- A change management and operational model that builds organizational readiness through training and tool improvements.
- A team approach, which helps deliver initial results in as little as 12 weeks, provides experiential learning, and enables independent, lasting customer success.

- A composable application architecture based on microservices, APIs, reusable components, and containerization.
- A scalable modernization roadmap for select strategic applications, which includes prescriptive guidance to work in a *split-and-seed* model. In this model, modernization capabilities and services are scaled across multiple engineering teams that focus on business outcomes. As new minimum viable products (MVPs) are defined, initial team members split up to create new product teams.

Modernization process

The goal of a phased approach to modernization is to provide incremental value by using modernization dimensions, to apply these to a subset of core, business-differentiating applications, and to accelerate modern technology adoption. The phased approach consists of three steps:

1. Evaluate the maturity of applications by using a modernization diagnostic playbook. Take a comprehensive approach to adoption, and develop processes that are aligned to intended business outcomes.
2. Start small and prepare to deliver an MVP to gain momentum by driving business results early and incrementally. This phase typically takes 12 to 16 weeks.
3. Create a scalable modernization roadmap to work in a split-and-seed model.

The following sections discuss these steps in more detail.

Topics

Step 1. Evaluate your applications

The goals of this phase are to:

- Thoroughly understand your application landscape and prepare your applications for modern data platforms, so you can accelerate the time to value without impacting your business, and then modernize, optimize, and scale.
- Profile your application landscape to identify the benefits, risks, and costs associated with change.
- Provide an end-to-end set of services: from strategy and planning; through deployment, migration, and application modernization; to ongoing support.
- Build policies, recommendations, and controls that provide reusable practices and tools to deliver ongoing business value.

In the evaluation phase, application owners and architects use a modernization diagnostic playbook to validate their modernization goals and priorities.

Using the modernization diagnostic playbook

A modernization diagnostic playbook provides a process for determining the value of moving from the current state to the future state for the enterprise. This is inclusive of technology changes that modernization involves.

You use the diagnostic playbook to determine the priority of your application or application suite for cloud modernization, and to identify the components that need to addressed during modernization.

Diagnostic dimensions

The modernization diagnostics playbook helps you understand the following dimensions of the current and target (post-migration) state of an application or a group of applications:

- Application grouping – Is there a reason to group applications (for example, by technology or operating model) for modernization?
- Sequencing – Is there an order in which applications should be modernized, based on dependencies?
- Technology – What are the technology categories (for example, middleware, database, messaging)?
- Dependencies – Do the applications have key dependencies on other systems or middleware?
- Environments – How many development, testing, and production environments are used?
- Storage – What are storage requirements (for example, the number of copies of the test data)?
- Operating model – Can all components of the application adopt a continuous integration and continuous delivery (CI/CD) pipeline?
  - If so, what infrastructure responsibilities should be distributed to application teams and to whom?
  - If not, what infrastructure responsibilities (for example, patching) should remain with a operations team?
- Delivery model:
  - Based on the application or group of applications, should you replatform, refactor, rewrite, or replace?
  - Which portion of the modernization should use cloud-native services?
- Skill sets – What expertise is required? For example:
  - A cloud application background to build applications with a modular architecture by using container and serverless technologies from the ground up.
  - DevOps expertise to develop solutions in the areas of CI/CD processes, infrastructure as code, and automation or application observability by using open-source and AWS tools and services.
- Modernization approach – Considering the current state of the applications, cloud technology choices, current technical debt, CI/CD, monitoring, skills, and operating model, what is the technical migration work that needs to be done?
- Modernization timing – What are the business portfolio timing considerations or other planned work considerations that might affect modernization timing?
- Unit and total cost of infrastructure – What is the annual cost of maintaining your workload on premises vs. on AWS, based on economic analysis?

Evaluating applications against these dimensions help you stay anchored in business, technology, and economics as you drive your modernization to the cloud.

Building blocks

When you're modernizing applications, you can classify your observations into three building blocks: business agility, organizational agility, and engineering effectiveness.

- **Business agility** – Practices that concern the effectiveness within the business to translate business needs into requirements. How responsive the delivery organization is to business requests, and how much control the business has in releasing functionality into production environments.
- **Organizational agility** – Practices that define delivery processes. Examples include agile methodology and DevOps ceremonies as well as role assignment and clarity, and overall collaboration and communication across the organization.
- **Engineering effectiveness** – Development practices related to quality assurance, testing, CI/CD, configuration management, application design, and source code management.

Identifying metrics

To learn if you are delivering what matters to your customers, you must implement measures that drive improvement and accelerate delivery. Goal, question, metric (GQM) provides an effective framework for ensuring that your measures meet these criteria. Use this framework to work back from your goals by following these steps:

1. Identify the goal or outcome that you are undertaking.
2. Derive the questions that must be answered to determine whether the goal is being met.
3. Decide what should or could be measured to answer the questions adequately. There are two categories of measures:
   - Product metrics, which ensure that you are delivering what matters to your customers.
   - Operational metrics, which ensure that you are improving your software delivery lifecycle.

Product metrics

Product metrics focus on business outcomes and should be established when the return on investment (ROI) for a new scope of work is determined. A useful technique for establishing a product metric is to ask what will change in the business when that new scope of work is implemented. It's helpful to formalize this thinking into the form of a test that focuses on what would be true when a modernization feature is delivered.

For example, if you believe that migrating transactions out of legacy systems will unlock new opportunities to onboard clients, what is the improvement? How much capacity has to be created to onboard the next client? How would a test be constructed to validate that outcome? For this scenario, your product metrics might include the following:

- Identify the business value test or hypothesis (for example, freeing *x* percent of transaction capacity will onboard *y* percent of new business).
- Establish the baseline (for example, the current capacity of *x* transactions supports *y* customers).
- Validate the outcome (for example, you have improved capacity by *x* percent, so can you now onboard *y* percent new business?)

Operational metrics

To determine whether you are improving your software delivery lifecycle and accelerating your migration, you must know your lead time and implementation time for delivering software. That is, how quickly can you convert a business need into functionality in production?

Useful operational metrics include:

- Lead time – How much time does it take for a scope of work to go from request to production?
- Cycle time – How long does it take to implement a scope of work, from start to finish?
- Deployment frequency – How often do you deploy changes to production?
- Time to restore service – How long does it take to recover from failure (measured as the mean time to repair or MTTR)?
- Change failure rate – What is the mean time between failures (MTBF)?

Step 2. Start small and build momentum

The goal of this step is to deliver an initial minimal viable product (MVP) to gain momentum. This approach enables you to drive business results early and incrementally.

Validating priority drivers

Before you start the modernization work with application teams, we recommend that you validate the priority drivers that you determined earlier. Follow these steps:

1. Compile the information you need from the diagnostic playbook.
   - Gather the priority drivers and feasibility assessment from the priority applications list.
   - Gather the transition and goal state dispositions for your applications.
   - Identify the application owners, architects, and stakeholders in cloud modernization planning.
   - Solicit information on dependencies or application suite sequencing, if known.
   - Determine how inventory entries relate to dependencies or application suite groupings. Applications might have individual components that are tightly coupled with, or dependent on, other components, and you might want to modernize these components together.
2. Schedule a one-hour or two-hour meeting with the people from step 1 to validate priority drivers.
   - Try to group multiple (up to three or four) applications by solution engineer or architect, and discuss them in one meeting, based on application dependency or application suite information.
   - Determine the roles and expectations for each team member for this upcoming meeting.
3. Conduct the meeting.

Finalizing details

After you follow the process in the previous section to validate the priority drivers, you can gather the details to determine the modernization approach and timing.

In this phase, the core team works side by side with application teams in short, two-day sprints to design a future state for their applications on the AWS Cloud. Activities include product definition, product discovery, story writing, value stream mapping, and designing CI/CD processes. Here are some ideas:

- Model each individual component of the application (for example, network configurations, storage configurations, databases, servers, and how the application is deployed on the servers).
- Deconstruct that model into its different building blocks and configurations by using tools such as containers or serverless technologies.
- Separate application functionality from any dependencies on underlying infrastructure. Abstract the functions of an application into components that you can move without changing any source code.
- Tightly integrate with DevOps by using CI/CD tools and mechanisms.

Building foundational platform services and modernizing applications

In this 12-week phase, the core team is supported by full-stack teams to deliver the prioritized business use case. This work is carried out by multiple two-pizza teams. For example, a platform engineering team is formed to develop foundational platform services, and a product team is formed to deliver new business outcomes:

- The platform engineering team configures, integrates, and customizes the AWS services that support the cloud foundation, developer workflow, and data analytics capabilities. Larger and more complex enterprises might have multiple teams supporting each of these capabilities.
- The product team develops new services and experiences for the business outcomes prioritized in the inception phase. As the product team develops new services, they also modernize core business capabilities.

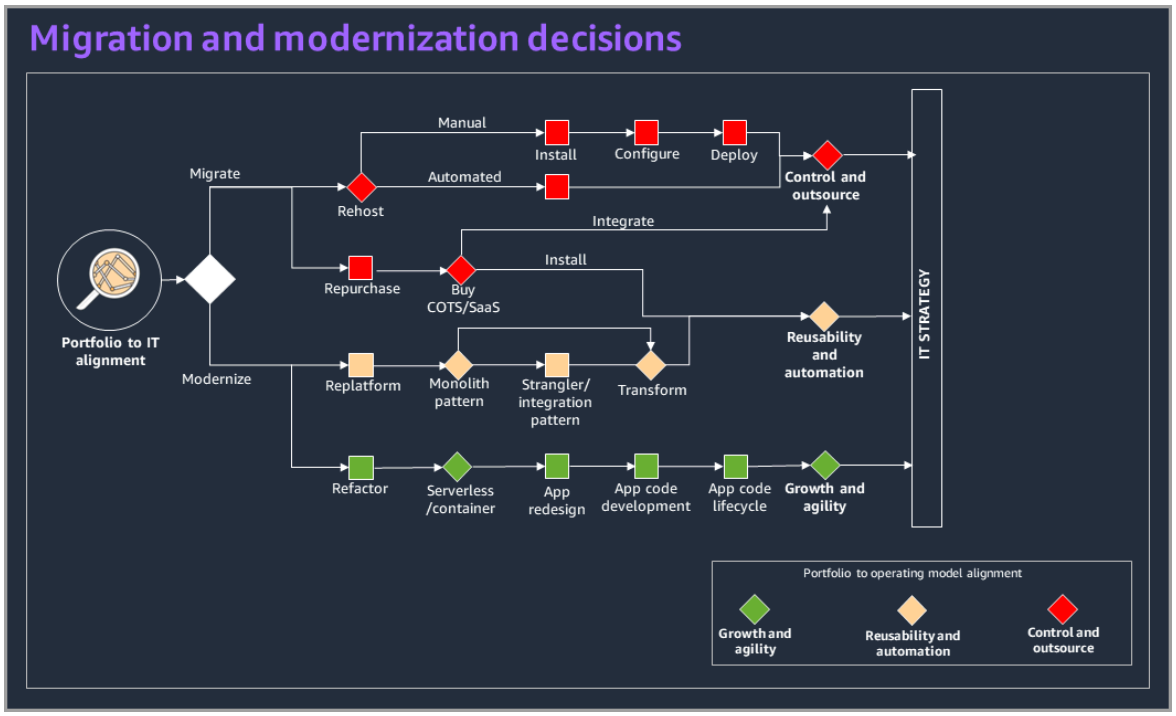
The platform engineering and product teams deliver a minimal viable product (MVP) that you can evaluate. Upon the success of the initial MVP, you can scale your modernization program by using a split-and-seed approach, whereby new applications are identified and initial team members are split up to create new product teams.

Step 3. Create a scalable modernization roadmap

After the initial MVP release of the prioritized outcomes and applications, we recommend that you develop a roadmap for scaling and accelerating your modernization efforts, improving developer productivity, and innovating rapidly. The core team splits and seeds new teams in order to scale your organization's capabilities and services across multiple engineering teams that are focused on business outcomes. By employing the split-and-seed approach over time, your organization can take on more development and accelerate the velocity of modernization.

The modernization roadmap should outline a pragmatic and continuous approach to application modernization with clearly defined patterns such as event-driven, strangler, domain-driven designs, decomposition, modern database options, and so on.

The roadmap should include a decision tree matrix, as shown in the following diagram, to identify a component of an application and move it to a managed service (such as a database service) with no changes to business logic, or to make code-level changes to improve performance, scalability, manageability, reliability, and resource usage.



Modernization readiness factors

Observe the following standards and best practices when you're modernizing your applications.

Topics

Code

- Provide code comments that document the functionality of your software, and use them to generate documentation.
- Follow code management and deployment processes that support frequent code check-ins and traceability to feature requests.
- Build test suites that include unit, functional, performance, and critical path tests, with 100 percent code coverage.
- Encourage code reuse to deliver the same or similar functionality in your code base.
- Develop prototypes to validate features with users before investing in full code development.

Build and test

- Redefine feature completeness based on testing, to improve quality and prevent recurring issues.
- Automate acceptance tests.
- Monitor all automated tests, and establish a process for handling failures in place.
- Track performance in both production and non-production environments, define service-level objectives (SLOs) based on realistic traffic and load testing, and provide the ability to scale to meet performance requirements.
- Abstract sensitive data from configuration files, and provide tools that automate and monitor configurations.

Release

- Automate deployments with support for dependencies (for example, database releases), regression testing, and tracking.
- Release code to the production environment incrementally, after every successful build.
- Manage feature flags (toggles) effectively: support run-time configuration, monitor usage, maintain flags throughout the development cycle, and assign owners by category.
- Provide traceability in your build pipelines, to track triggers, failure notifications, and successful completion.

- Run automated deployment processes and tests for “zero touch” code updates in continuous delivery.
- Use zero-downtime, fully automated blue/green deployment methodologies.
- Make sure that your database schema changes are implemented consistently across all development and production environments.

Operate

- Create a DevOps triage runbook that’s integrated with your notification system.
- Make sure that your monitoring and notification system meets service-level objectives (SLOs) and supports thresholds, health checks, non-standard HTTP responses, and unexpected results.
- Establish effective risk management and disaster recovery processes.
- Develop a log rotation and retention strategy that meets your business and legal requirements.
- Develop dashboards that track product performance, measure the success of new features, and display alerts when metrics don’t meet expectations.

Optimize

- Review and improve processes regularly, based on performance and quality measures.
- Implement root cause analysis and prevention processes to prevent issues from recurring.
- Provide data-driven metrics that capture product health, and make sure that all notifications and actions are based on these metrics.

Readiness

- Dedicate a cross-functional team (including business partners, developers, testers, and architects) to your modernization efforts.

Next steps

This guide provided a phased and incremental approach to modernizing your applications. We recommend that you begin scoping your workstreams with technology and delivery for the short, medium, and long term. You can expect to invest more in tools, frameworks, and practices to drive more effective delivery and engineering practices across teams. Set goals for the modernization process, understand each application you're planning to modernize, and choose the optimal modernization approach for each application. Monitor and optimize your progress.

AWS will engage with your organization's business development teams to drive technology modernization and to manage your end-to-end needs, including application architecture, design, and development; mobile enablement; testing; and collaborative solutions. These services combine proven processes, intelligent automation, use of data and patterns, open standards, and the right people to help you modernize your legacy applications. Primary goals are targeted to:

- Assist to develop a platform that offers a fully automated, tools-based approach for modernizing legacy technology, and couple it with holistic knowledge of modernization.
- Build an MVP with full stack teams, with modernization engineering capabilities focused on customer outcomes and scale.

AWS Professional Services and AWS Partners work directly with senior technology and business customer stakeholders to drive enterprise modernization, while iteratively delivering end-customer value.

Resources

Related guides

- Strategy for modernizing applications in the AWS Cloud
- Evaluating modernization readiness for applications in the AWS Cloud
- Modernizing operations in the AWS Cloud
- Prescriptive guidance for migrating to the AWS Cloud
- Decomposing monoliths into microservices
- Integrating microservices by using AWS serverless services
- Enabling data persistence in microservices

AWS resources

- AWS documentation
- AWS general reference
- AWS glossary

Document history

The following table describes significant changes to this guide. If you want to be notified about future updates, you can subscribe to an RSS feed.

Change	Description	Date
Initial publication (p. 12)	—	December 18, 2020

AWS Prescriptive Guidance glossary

The following are commonly used terms in strategies, guides, and patterns provided by AWS Prescriptive Guidance. To suggest entries, please use the **Provide feedback** link at the end of the glossary.

Modernization terms

business capability

What a business does to generate value (for example, sales, customer service, or marketing). Microservices architectures and development decisions can be driven by business capabilities. For more information, see the Organized around business capabilities section of the Running containerized microservices on AWS whitepaper.

domain-driven design

An approach to developing a complex software system by connecting its components to evolving domains, or core business goals, that each component serves. This concept was introduced by Eric Evans in his book, *Domain-Driven Design: Tackling Complexity in the Heart of Software* (Boston: Addison-Wesley Professional, 2003). For information about how you can use domain-driven design with the strangler fig pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

microservice

A small, independent service that communicates over well-defined APIs and is typically owned by small, self-contained teams. For example, an insurance system might include microservices that map to business capabilities, such as sales or marketing, or subdomains, such as purchasing, claims, or analytics. The benefits of microservices include agility, flexible scaling, easy deployment, reusable code, and resilience. For more information, see Integrating microservices by using AWS serverless services.

microservices architecture

An approach to building an application with independent components that run each application process as a microservice. These microservices communicate through a well-defined interface by using lightweight APIs. Each microservice in this architecture can be updated, deployed, and scaled to meet demand for specific functions of an application. For more information, see Implementing microservices on AWS.

modernization

Transforming an outdated (legacy or monolithic) application and its infrastructure into an agile, elastic, and highly available system in the cloud to reduce costs, gain efficiencies, and take advantage of innovations. For more information, see Strategy for modernizing applications in the AWS Cloud.

modernization readiness assessment

An evaluation that helps determine the modernization readiness of an organization's applications; identifies benefits, risks, and dependencies; and determines how well the organization can support the future state of those applications. The outcome of the assessment is a blueprint of the target architecture, a roadmap that details development phases and milestones for the modernization process, and an action plan for addressing identified gaps. For more information, see Evaluating modernization readiness for applications in the AWS Cloud.

monolithic applications (monoliths)

: Applications that run as a single service with tightly coupled processes. Monolithic applications have several drawbacks. If one application feature experiences a spike in demand, the entire architecture must be scaled. Adding or improving a monolithic application's features also becomes more complex when the code base grows. To address these issues, you can use a microservices architecture. For more information, see Decomposing monoliths into microservices.

polyglot persistence

: Independently choosing a microservice's data storage technology based on data access patterns and other requirements. If your microservices have the same data storage technology, they can encounter implementation challenges or experience poor performance. Microservices are more easily implemented and achieve better performance and scalability if they use the data store best adapted to their requirements. For more information, see Enabling data persistence in microservices.

split-and-seed model

: A pattern for scaling and accelerating modernization projects. As new features and product releases are defined, the core team splits up to create new product teams. This helps scale your organization's capabilities and services, improves developer productivity, and supports rapid innovation. For more information, see Phased approach to modernizing applications in the AWS Cloud.

strangler fig pattern

: An approach to modernizing monolithic systems by incrementally rewriting and replacing system functionality until the legacy system can be decommissioned. This pattern uses the analogy of a fig vine that grows into an established tree and eventually overcomes and replaces its host. The pattern was introduced by Martin Fowler as a way to manage risk when rewriting monolithic systems. For an example of how to apply this pattern, see Modernizing legacy Microsoft ASP.NET (ASMX) web services incrementally by using containers and Amazon API Gateway.

two-pizza team

: A small DevOps team that you can feed with two pizzas. A two-pizza team size ensures the best possible opportunity for collaboration in software development. For more information, see the Two-pizza team section of the Introduction to DevOps on AWS whitepaper.