aws startups

Fundamentals of building your MVP

Introducing MVPs

So you want to launch a startup. You have a great idea and friends that want to help. First step, you will want to test out your hypothesis. Good startups come from identifying problems or emerging opportunities and executing on them.

But getting that great idea built and into the market is more likely to succeed with an MVP, or **minimum viable product.**

In the early stage, a startup's primary goal is product market fit, and building an MVP is the best way to do that. You define the minimum set of features, get it into users' hands, and iterate on their feedback.

An MVP is the first product that you build. It helps you understand how people use your product and how it fits into the market. With an MVP, you get to market faster, get feedback sooner, and continuously fine-tune your product as you learn more about your market and customers.

So what exactly is an MVP? The meaning is in the name: Minimum Viable Product.

Minimum means keeping a tight focus on the smallest unit of functionality you can test with users. Think of this as a stripped-down, essential version of your product.

You'll be tempted to add details and build in new features. Resist the urge! Focus on the minimum feature set, the leanest possible version of your product.

It may feel uncomfortable to release an MVP. Frankly, it should. Your product is still incomplete, and you're working with users to make it better for them.

"If you aren't embarrassed by the first version of your product, you shipped too late."

- Reid Hoffman, Co-Founder LinkedIn

What makes an MVP "Viable”?

Viable means that you’re testing just enough features to make the product useful for early adopters and beta-testers. The wider market can wait.

Viability is a catch-all term for multiple aspects of the product experience.

Usability describes how easy it is to get the product into users’ hands and gather feedback. No “nice-to-haves” or spiffy design; stick to core, must-have features.

Lovability expresses users’ desire for a better experience than existing products offer

Testability determines whether there are risky parts to your business model.

Product, simply put, is the object or service that you put in your customers’ hands.

Your MVP may feel puny compared to your vision of the product at scale. That’s ok. Your MVP should showcase your product’s essential function, no more. Simple and pared-down doesn’t mean non-functional however. **Your MVP has to work before you share it with users.**

Aim to build an MVP that achieves the immediate need. This means building end-to-end for a specific user, rather than myriad features for all users. Over the fullness of time, you can roll out additional features. Early adopters will get incremental benefits with each release, and your product will become more appealing to different types of users.

Why Build an MVP?

Startups have to juggle competing time pressures. On one hand, there's the desire to move quickly and innovate with speed. On the other, there's your bottom-line commitment to creating products that solve real problems.

Startups limit risk by testing hypotheses with an MVP. Constant user feedback via analytics or direct observation guides your investment of time and talent in areas users favor, or find problematic. This helps you focus on areas of customer interest and design in response to market forces.

"Invention requires two things: the ability to try a lot of experiments, and not having to live with the collateral damage of failed experiments."

- Andy Jassy, CEO, Amazon Web Services

Over time, MVPs give product teams valuable feedback on what users enjoy, use most often, or ignore entirely. In this way, MVPs can help you move from imagining how people might use a product to understanding how it actually works in someone's hands. It also saves significant engineering time that would otherwise be wasted building unnecessary features.

MVP Vs. Prototype

While prototypes seem similar to MVPs due to their iterative nature, MVPs fulfill an entirely different function than prototypes.

Prototypes are tightly-controlled experiments before a product finds its way into users' hands. They describe how various components fit together to create a product. Prototypes help you and your team understand how your product works before users are invited to "test-drive" it.

An MVP is an outward-facing tool to test product-market fit with actual users. MVPs help startups answer questions such as what do customers want and what will users use. It is the first opportunity users have to interact with your product.

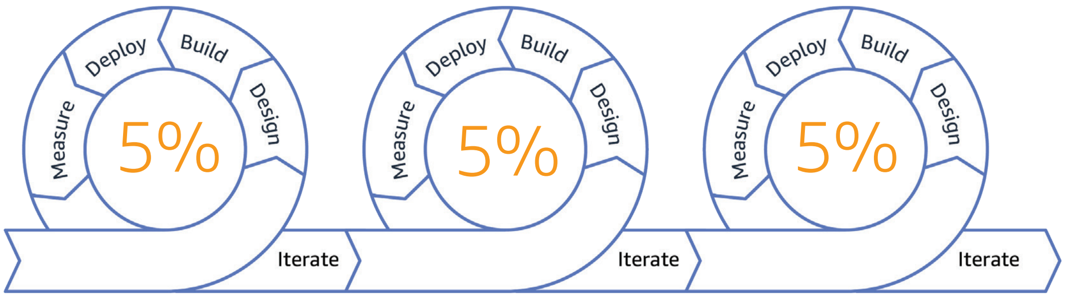
Another point to emphasize is that your MVP should be the SIMPLEST and most BASIC product that can test your hypothesis. For example, consider ride-sharing apps. The developers' hypothesis is that there is a need for peer-to-peer driving. Creating a rideshare MVP means outlining the end-to-end user experience and defining the primary users. Accordingly, the ride-sharing MVP offered both drivers and passengers a simple and easy way to connect.

Those same apps today offer all kinds of features to customize the rider experience. None of those would be useful if the hypothesis was false and the product useless. But once the MVP proved the core hypothesis correct, the app could evolve to offer additional features.

Building Your MVP

Each MVP is unique, as it represents your unique product. Yet the process of building any MVP is the same in terms of the development cycle and common anti-patterns that emerge.

The Golden Rule of development is constant iteration, which allows your team to move quickly and incrementally improve the product. The aim of each iteration is to incrementally improve your product, for example, gaining 5% improvement with each iteration. Within each iteration are four sequential steps:
Design -> Build -> Deploy -> Measure.



Before diving in, a word of caution: **Don't overbuild!** Simple solutions are better than overbuilt ones. MVPs are never perfect, so avoid getting caught up in edge cases and over-engineering.

Once the MVP starts proving your hypothesis, you can add features to optimize and enrich the user experience. Consider our ride-share app example from before. The MVP simply connected riders with drivers and cars. Once proven, you can add features such as split fares with friends and book future rides. These features were informed by user experience and feedback.

As you plan your MVP, organize the work ahead by drafting a working roadmap to document and rank your priorities. Roadmaps are never static documents though, they will change and evolve based on the feedback you collect from customers.

Start a Sprint

In development, a "sprint" is a preset chunk of work that can be completed in a set period of time. Sprints are short, intense bursts of work that start with advance planning and end with a retrospective evaluation. Sprints that are designed well end with a feature that can be deployed and generate user feedback.

Complete

Uninterrupted

Short

Be careful not to make sprints too long. This introduces the risk of interruptions and changes before development is complete and feedback is gathered. The time for feedback is after one sprint finishes and before the next begins. The shorter the sprint, the faster you get feedback and iterate.

When prioritizing work in a sprint, consider speed and impact. Ask yourself if the work could be done by a single developer in a single sprint and if the work affects metrics. Start with features that are high-impact and can be completed quickly. If the feature is too complex, break it into smaller micro-tasks small enough for one engineer to accomplish within a single sprint. This also makes the feature easier to deliver, test and validate with users.

Performing Standups

Running sprints and iterating quickly often requires more coordination within and across teams. Standups are fast, informal group check-ins run on a daily basis that foster collaboration, surface problems, and keep the team aligned to plan.

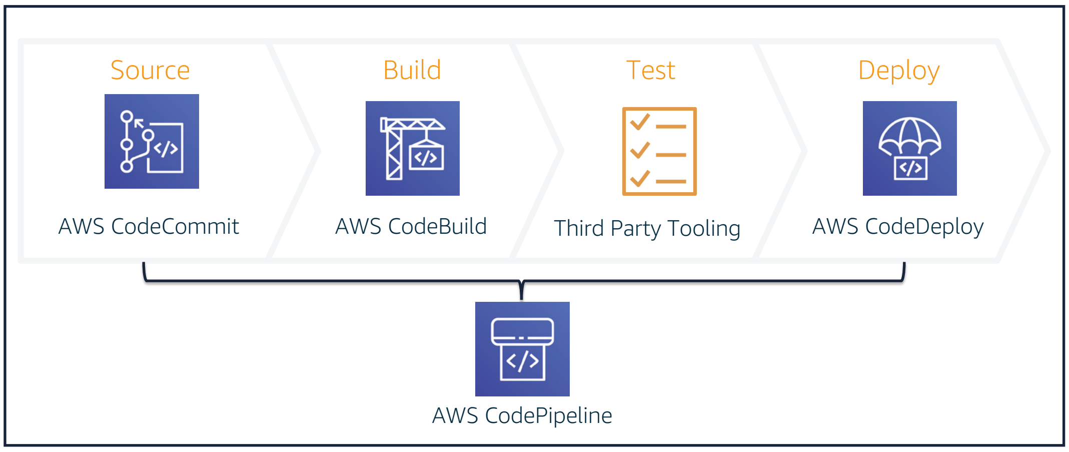
Key Elements of Standups Include:

- **All-hands-on-deck.** Everyone should be there: technical, product, marketing & business. Often, tech is blocked by something that another team can resolve with clarification.
- **Speed is of the essence.** Everyone gets one minute, and no more, to say what they did yesterday, what they'll work on today, and where they're stuck or blocked.
- **Cut off at 15 minutes.** Aim to resolve issues as quickly as possible or take offline. If you often run over 15 minutes, your team is either too big or the updates aren't concise.
- **Include remote staff.** Be sure to include remote teams or people working from home in standups via video call. If time zones pose a challenge, use written updates.

Continuous Delivery

Many development teams have adopted Continuous Delivery (CD) as a means to continuously roll out features that are developed in each sprint. This keeps your product updated with the latest features and helps build user momentum.

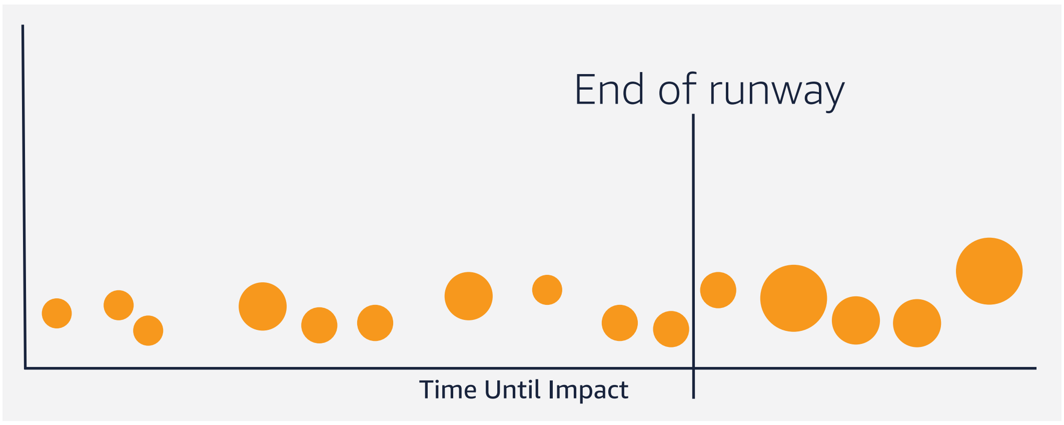
Ideally your continuous delivery toolchain should be tightly integrated with your development environment. AWS CodeStar for example creates the toolchain in minutes and provides a unified interface for software development in one place. This enables greater automation and shortens delivery times for releasing new features.



Navigating Anti-patterns

No matter how well you've prepared, you're going to hit roadblocks. One way to predict a possible roadblock is by recognizing anti-patterns, common responses to problems that turn out to be unhelpful or counterproductive.

A common problem for startups is building features that will be needed too far into the future. This approach doesn't work because the future is not guided by user feedback and the planning required for such features is more extensive. This means investing time, effort, and money that might never return value or would only have impact well after your startup runs out of funding.

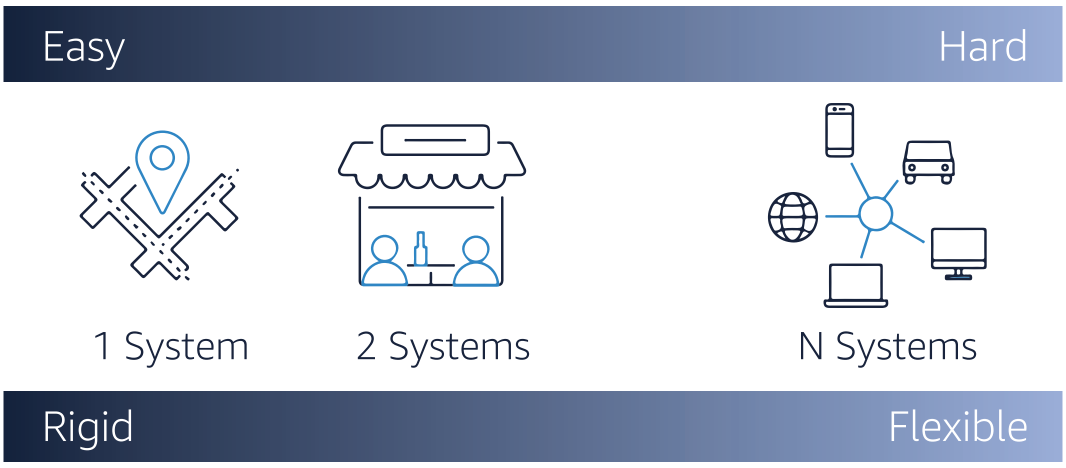


In the above graph each orange dot represents a feature. The bigger the dot, the longer it takes to build. The fast pace of startups means things you build now may never return value.

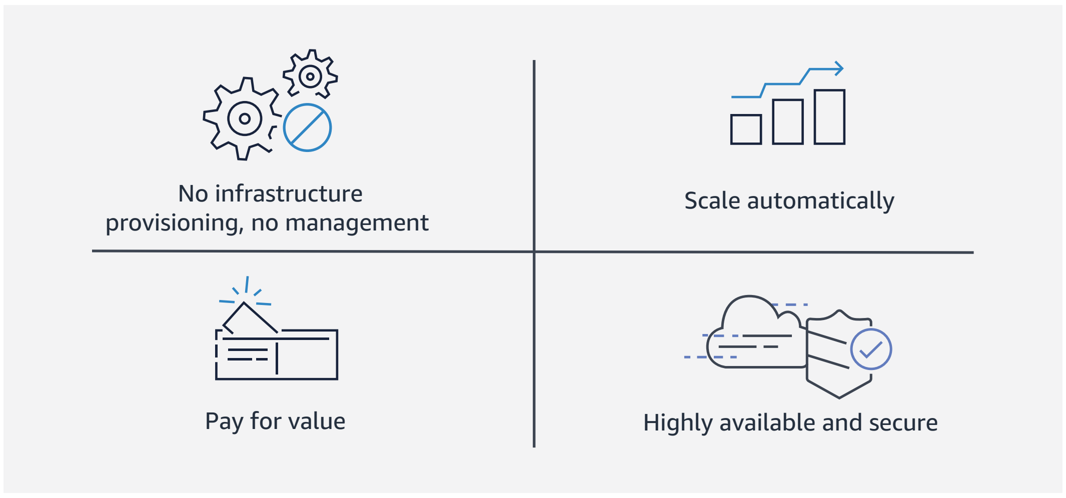
Technical debt is another inevitable challenge of the development process. Tech debt accrues when development prioritizes speed of implementation over technical "purity". For example, code you wrote when your goal was X is no longer optimal, because now, your goal has shifted to Y. The time and effort that is required to rewrite this code is technical debt, time better spent by engineers working on features.

Technical debt does not necessarily mean bad or inefficient engineering. It is simply the cost of moving quickly and delivering features. Understanding that technical debt is part of the MVP development process is key to tracking your progress against the roadmap.

One way to reduce technical debt is to generalize features that might support more needs in the future. By abstracting code to cover multiple uses cases, it provides flexibility and avoids the need to refactor code later. Case in point, let's say you have an app that takes card payments. It makes sense to generalize this feature to also accept other forms of payment. Whenever you opt to include subscription or additional payment methods, it's easier to do so having created an abstraction to cover that use.



The last anti-pattern to highlight is building something from scratch that already exists. While it is exciting to build new things, maintaining this code over time has drawbacks. Everything you build has a cost. Using cloud infrastructure provided by AWS, implementing microservices, adopting open source, and leveraging serverless are some options to avoid the time and cost on undifferentiated heavy lifting, freeing you up to deliver more valuable features to users.


No infrastructure provisioning, no management
Scale automatically
Pay for value
Highly available and secure

Conclusion

We hope this guide has clarified the purpose and potential power of building and launching an MVP. It's a complex process that's essential to product development, but know that you don't need to go it alone and AWS can help!

Scan here to watch an **AWS Startup advocate talk about building an MVP**