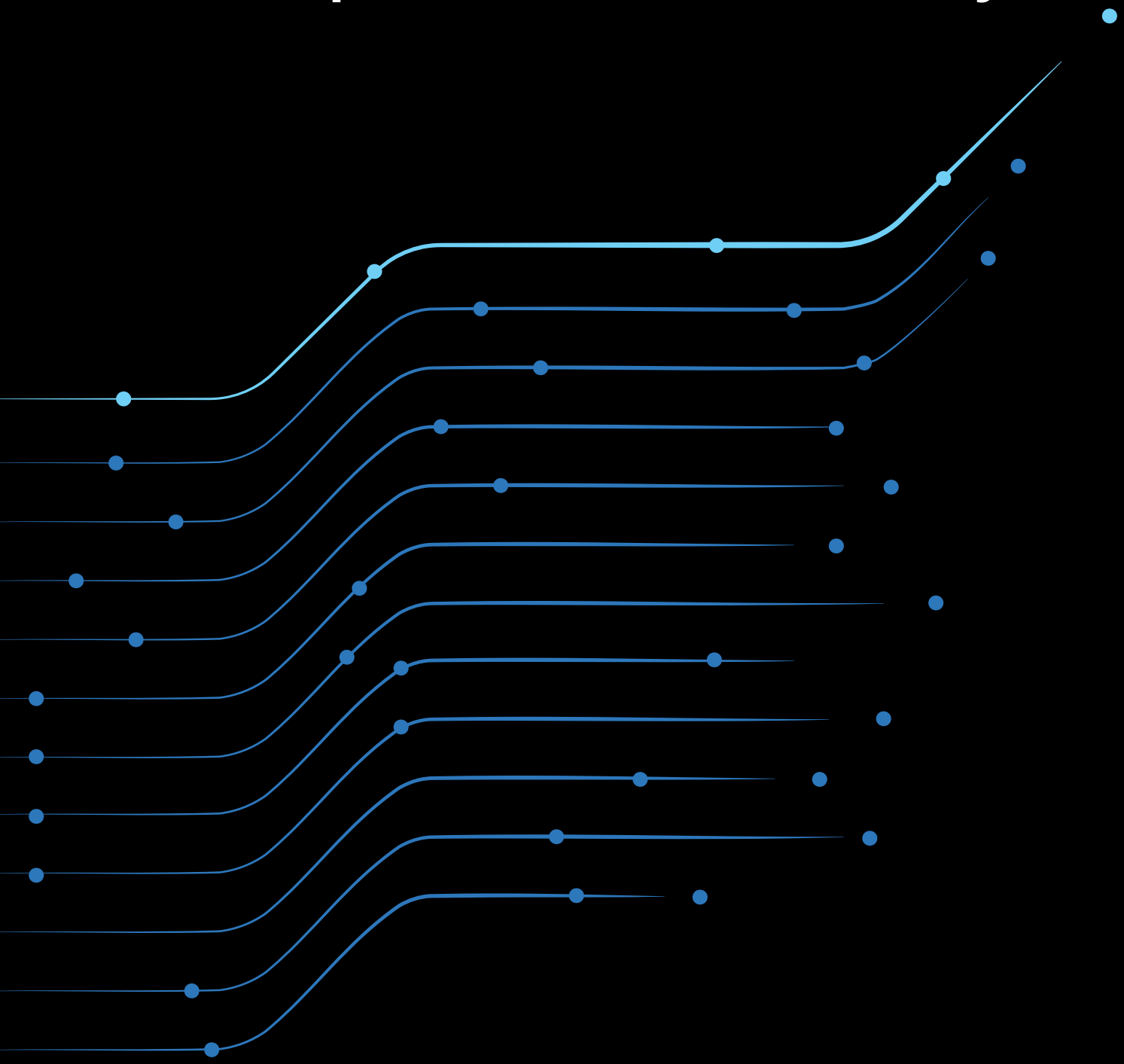
Microsoft

7 Tips to Optimize Remote Developer Team Productivity



tl;dr

Research shows that developer teams play a key role in driving business performance.[1] That's why empowering them to maximize their potential is such an important focus area for leaders. But that job has become more complicated today as organizations embrace remote work and team leaders are increasingly managing fully distributed teams. How can technical leaders enable these distributed developer teams to build productively, collaborate efficiently and securely, and drive innovation at scale, regardless of location?

The tips inside include how to:

- Maintain your team's culture in an online environment.
- Provide your developers with processes and tools to help them code remotely, on demand, while enhancing team collaboration.
- Apply automation principles and track application health across the entire development lifecycle in a remote, distributed organization.

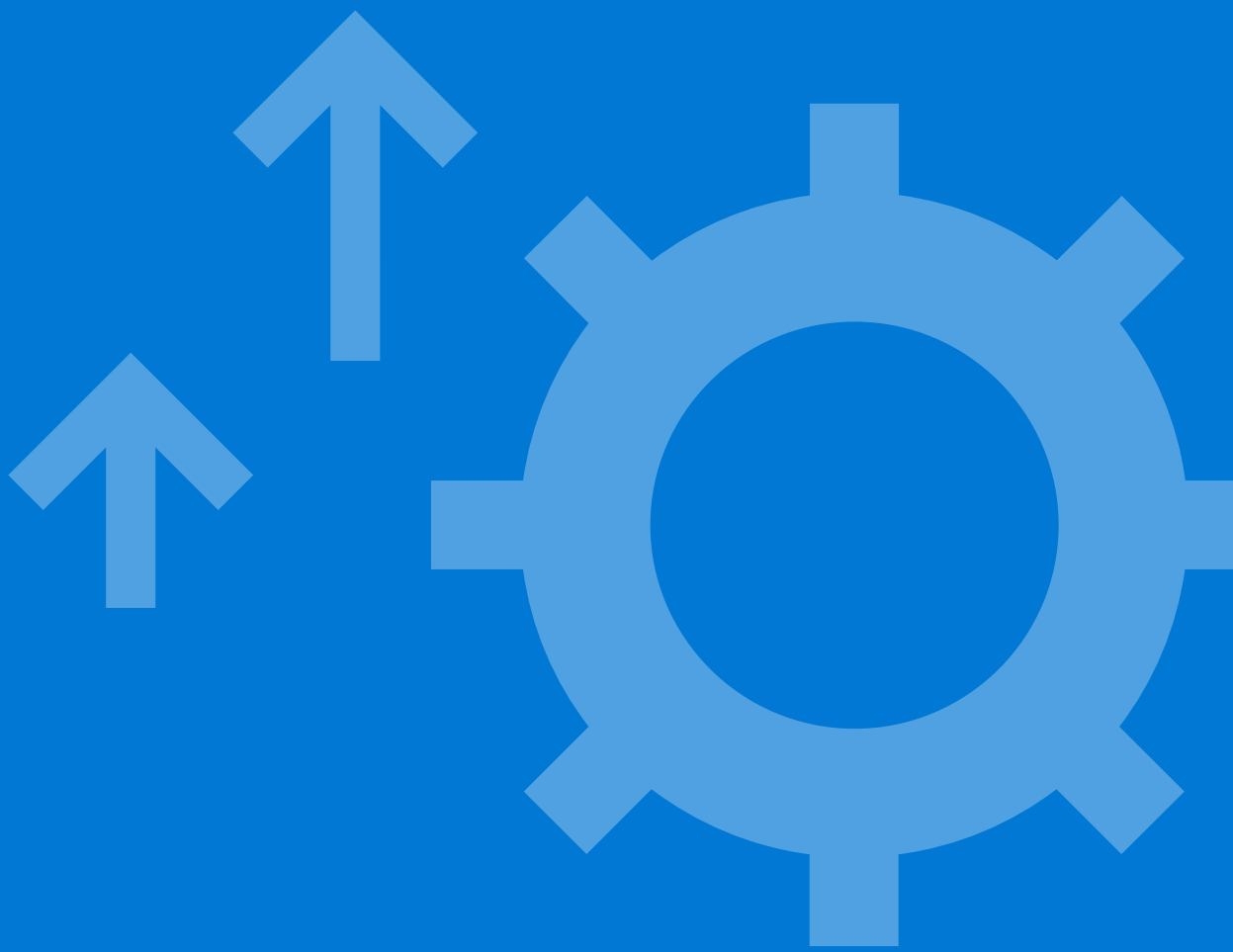
[1] "Developer Velocity: How software excellence fuels business performance," McKinsey, 2020.

This book is for those who...

- Coordinate or lead a group of developers across a distributed or partially distributed team.
- Wonder what slows developers down when they're working and collaborating remotely.
- Have a role like software architect, engineering manager, development manager, or technical lead.
- Want to be a more effective technical leader.
- Want to understand some of the common challenges of distributed developer teams and how you can improve areas like productivity, collaboration, and culture.

Table of contents

7 tips to optimize remote developer team productivity



Introduction

Research by McKinsey[1] has shown a strong correlation between the degree to which leaders empower their developer teams and resulting business performance. The findings illustrate that the more you enable developers to build on their terms and collaborate globally and securely—whether remotely or co-located—the better the business outcomes tend to be in terms of performance, innovation, customer experience, brand, and talent growth. But as organizations increasingly embrace remote working across their employee base, what can developer teams do to continue to optimize their productivity, agility, and collaboration? As a technical leader, how can you help ensure your distributed developer team can code efficiently, collaborate seamlessly, and ship securely from any location?

Examination of the same McKinsey report reveals that the number one driver of business performance is best-in-class developer tools. Organizations with strong tools are 65 percent more innovative, and their developer satisfaction and retention rates are 47 percent higher. Equipping your developer teams with the right tools, skills, and culture for remote development is one of the most effective ways to contribute to delivering these business outcomes.

The good news is that developers as a whole were actually early pioneers when it comes to effective remote working, establishing vast communities, like GitHub, based largely on the promise of improving collaboration between distributed contributors. With many of your developers already skilled at remote collaboration, you may be surprised by the gains your team can achieve by simply enabling them to harness that skillset within the context of distributed professional work. Here are seven tips on how, as a technical leader, you can deliver just that.

[1] "Developer Velocity: How software excellence fuels business performance," McKinsey, 2020.

Tip 1

Take your developer culture online

Building and maintaining a strong developer culture is one of the most important tasks for a technical leader.

This can be a challenge when everyone is co-located, but it can be even harder when the people on your team are physically apart. However, with the right leadership and tooling, your remote developer team culture can thrive.

If you've ever worked in a remote role, you've probably experienced some of the challenges it can create in terms of not feeling connected to individuals on your team or plugged into a meaningful culture. People tend to have a strong social response to routines and rituals, casual conversations, and after-hours gatherings that naturally arise within co-located environments. Without those interactions, a healthy team culture can be difficult to establish because every touch point is work-focused. Team members run the risk of never really getting to know one another, and they may not be able to reach a comfort level where communication flourishes. This can be a much bigger problem than people realize.

Start with cultivating a healthy culture

Enabling your team to show who they are, what's important to them, or even just their sense of humor can improve collaboration and productivity by developing better team communication and a deeper understanding of everyone's capabilities and potential. It can also help with attracting and retaining talent. But where do you start?

Since those interactions won't come out naturally in a virtual environment, consider creating a dedicated space and time to encourage and facilitate this type of sharing and communication amongst your team. Today's leading collaboration platforms, such as Microsoft Teams, offer a range of capabilities that can support this. For example, you can create chat channels that act as a virtual watercooler for people to connect informally in whatever way they want. That could be sharing memes, talking about their hobbies, their latest TV obsession, or an interesting article on development practices they read. Some even go as far as enabling automated 'icebreaker bots' to create "coffee time" between team members.

You can create chat channels that act as a "virtual watercooler" for people to connect informally.

Beyond a virtual watercooler, and generally working toward facilitating richer and more personal interactions across the team, as a leader you can build culture by using additional processes and resources to demonstrate empathy, contribute to individual self-development, and provide a safety net for individuals who run into remote working challenges. For example, consider a revised, more deliberate approach to your one-on-one time with developers.

Tip 1: Take your developer culture online

Routine check-ins are important regardless of location, as they help ensure people feel appreciated and are empowered from a career growth perspective. But you'll likely need to adjust the cadence of these check-ins to compensate for less overall engagement, and you'll also likely need to add more structure to the meetings to help increase personal sharing and feedback, especially when it comes to their productivity and well-being.

You may also consider giving your team the option of participating in wellness or learning sessions that specifically tackle common remote working issues. Individuals often won't proactively reach out if they're having trouble maintaining productivity or feel like they're unable to learn as quickly in a remote environment. Short sessions and even online resources that offer practical tools like "productivity tips for the (basement) office commute" will go a long way toward catching these issues while they're outside your daily purview and before they become a larger problem.

Remember: a happy team is a productive team. Creating a strong culture online will not only increase performance, it will help with talent, too. Top performers naturally gravitate to companies that have built up this reputation.

Think through the tools you need to bring your culture to life

Today's tools can enable you to create virtual environments that facilitate healthy culture development across distributed teams, and those same tools can be used to bring your team together in a way that feels much closer to a co-located experience.

Three foundational capabilities for a strong virtual team culture:

1. Sustain rich, seamless, real-time communication with instant messaging, audio calling, and video conferencing

2. Provide a single source of truth with group storage and file sharing

3. Create topical discussions and enhance team collaboration with tailored virtual workspaces

For example, Microsoft Teams offers your team the ability to chat, meet, call, and collaborate from anywhere, and from any device. Communication is enriched and accelerated as people have the option to pick the right communications mode for the task, shifting away from transactional communications channels like email for tasks that need more agile, real-time collaboration.

The combination of video conferencing, instant messaging, screen sharing, and developer tool integration enable your distributed team to hold daily stand-ups and scrums as if they were in the office. You can even seamlessly bring back activities like whiteboarding as a team via integration between Microsoft Teams and Microsoft Whiteboard.

Looking beyond individual meetings and moments in time, having a persistent collaborative space your developers can maintain with knowledge sharing and progress updates will also enable them to better track overall project status and other deliverables. Based on the work that needs to be done, they can communicate via a visual model, such as a virtual Kanban board on a Microsoft Teams' Wiki tab—enabling them to recreate this essential tool in a virtual environment. It's also possible to integrate many developer tools into Microsoft Teams, such as Azure Boards, for a variety of comprehensive solutions aimed at enhancing collaboration and team management across key workflows.

Even activities like whiteboarding, are now available via Microsoft Teams.

For more information on how you can maximize the value of Microsoft Teams, from collaboration to scrums and boards to useful integrations and workflow enhancements, Sid Uppal, one of the founding members of Microsoft Teams, put together an in-depth DevOps team example here.

To illustrate how cultivating a strong developer team culture online can look in practice, let's look at the fictional story of Amy the tech lead.

Amy's story

Cultivating a team culture remotely

Amy sat down at her desk and logged into her work laptop with a hot cup of green tea at her side to start the day. She scanned her developer team's virtual Kanban board, noting most of the cards had progressed. As the Tech Lead for the Customer Onboarding team, she was happy to see no obvious blockers. She skimmed through her email before opening the General chat channel in Microsoft Teams. This channel functioned as the main room for her fully distributed team.

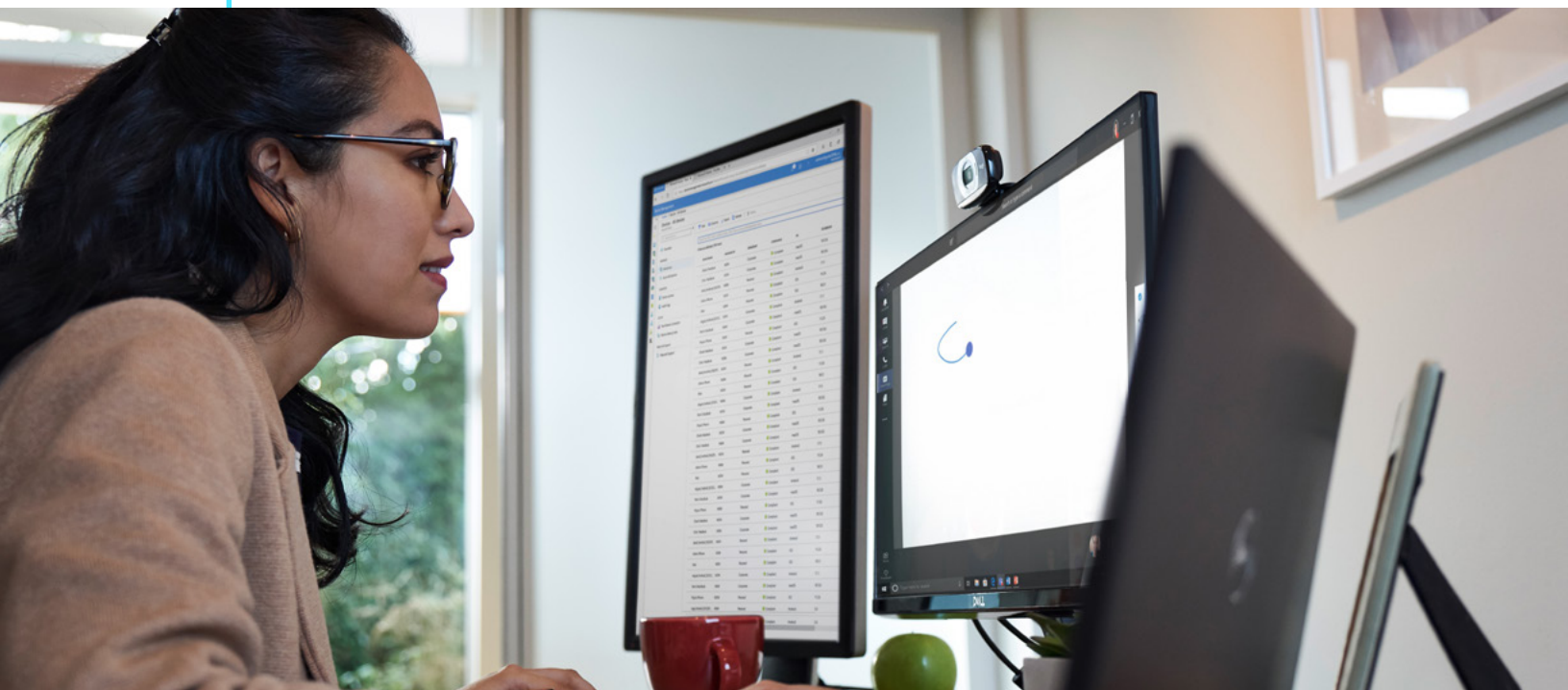
Amy scrolled back through the messages her team had posted during the night. She saw a few updates and requests for help. She noticed Zoe respond to Jay's request for help. Amy wrote herself a reminder to recognize this action in her one-on-one with Zoe later that day. To build a strong team, Amy knew to encourage team members to ask for and provide help to each other.

Amy continued scrolling through the messages. She paused, smiling at suggestions to celebrate team accomplishments for the month. When working with a co-located team, Amy would default to buying celebratory treats, but with everyone in different cities and time zones, this approach wouldn't work. One person suggested offering verbal appreciations for team members during their end-of-month virtual meeting. Another suggested using Microsoft Whiteboard, their virtual team diagramming tool that's integrated into Microsoft Teams, to collectively draw an image. The single image would represent something meaningful to the team. Amy was always impressed by the creativity of her team. Amy picked five options she felt were most realistic and fun then created a poll in the team chat. She knew her developers disliked unnecessary meetings that could be replaced with automated decisions—this way they could vote and the result would be automatically posted the next day.

Tip 1: Take your developer culture online

Amy continued to scan through the chat. She laughed out loud at some memes posted, adding a few emoji reactions to contribute to the team's informal "Chit chat" channel. She also saw "Chris," the coffee-time bot, was hard at work. The automated icebreaker bot set up informal coffee-time chats between random team members. It sent calendar invites automatically, even suggesting a few questions as conversation starters. This week's theme seemed to be about favorite programming language features. Amy appreciated these automated nudges, knowing how important it was for team members to learn more about each other.

Amy also saw that Jay shared a link to his finished design document for integrating a new onboarding partner. After clicking on the link, Amy reviewed the proposal, scanning through comments from other team members and noting the final outcome. She thought Jay had landed on a strong technical solution. Returning to the channel, Amy posted a large thumbs up GIF in response, an informal team tradition to show her support.



Tip 2

Enable your team to code on-demand, from anywhere

Common productivity inhibitors and roadblocks, such as facilitating onboarding, environment setup, and providing required training can have an outsized impact on a distributed team.

Everything from troubleshooting issues to procuring new hardware and configuring new environments can become exponentially more difficult remotely. Should your highly skilled developers spend time and energy worrying about hardware, software, or location when they could be laser-focused on contributing to the success of the project?

Streamline the setup process for new developer environments

The most important step you can take to eliminate or at least mitigate the impact of these roadblocks when they inevitably arise is to streamline and automate the setup and onboarding process. You don't want your team dealing with the tedious details of setting up a virtual machine (VM) and code editor, or checking out code when they could be problem-solving or coding. Worse than this, new team members may have to wait until another team member is online before they can start or complete this process, which, depending on time zones, could cause further delays.

One way to start is by providing VM images that your team can use to quickly set up new local development environments. Ensure these VMs are kept up-to-date with the team's favorite tools installed and preconfigured, including setting up code editors with team coding conventions. It's also a good idea to invest in a script that automates the project setup given an empty environment. Example tasks might include checking out code, setting up a default database, or creating a local configuration file for development. By streamlining and automating these steps you'll not only dramatically reduce downtime in the event of some major hardware or software issue, but you'll also enable team members to rapidly create new local development environments wherever they are, providing extreme flexibility to relocate or use multiple machines.

Enable your team to spin up pre-configured developer environments in just a few clicks

While streamlining and automating local setup has its benefits, a more modern approach is to use new cloud-hosted environments and tools that can dramatically speed up this process for your team.

Even when using VMs, developers often spend endless hours configuring environments for new projects—just to do it all over again once that project is complete. Visual Studio Codespaces brings your development environment into the cloud, removing local dependencies and manual configuration. Developers can set up fully configured cloud-hosted development environments in just a few clicks. Codespaces' cloud-hosted environments are then accessible in minutes from any device via a browser-based editor, making it a powerful resource for distributed or mobile teams that can't depend on routine access to a single location or machine.

Developers can set up fully configured cloud-hosted development environments in just a few clicks.

And Visual Studio Codespaces isn't limited by its simplicity and ease-of-access—its environments are fully-configured with all the tools developers need to be productive for any type of project. They're fast to create and disposable, featuring Git repo support, extensions, and a built-in command line interface where your developers can edit, run, and debug applications.

Tip 2: Enable your team to code on-demand, from anywhere

One final point to keep in mind is that streamlining the local development environment setup process or removing it entirely with cloud-hosted environments will do more than just help your existing team. You'll also automate the onboarding process for new team members, which will keep their energy high by letting them avoid the boring stuff right when they are most excited to dive into the work you hired them to do. Pair automated onboarding with strong team checklists and resources and they'll be able to immediately jump into tasks like making small changes to code, so they feel productive, along with dedicating more time to learning about the product, tools, and technologies via free, interactive learning courses, such as those on Microsoft Learn.

To illustrate how the right on-demand tools and processes can enhance distributed developer productivity in practice, let's look at the fictional story of Ryan the remote developer.

Ryan's story:

Seamless developer productivity from anywhere

Today is an exciting day for Ryan, who works remotely on the Customer Onboarding team. After two years of work and numerous big projects completed, it's finally time to retire his trusty old laptop—he's expecting to receive a new Surface Book 3 to replace his current machine.

Ryan commits and pushes his last code change on his old laptop before getting his lunch. He glances at the CI build before going upstairs to his kitchen, noting that while it looks set for completion, he'll get a notification via Microsoft Teams on his phone if his change has created an issue with the team build.

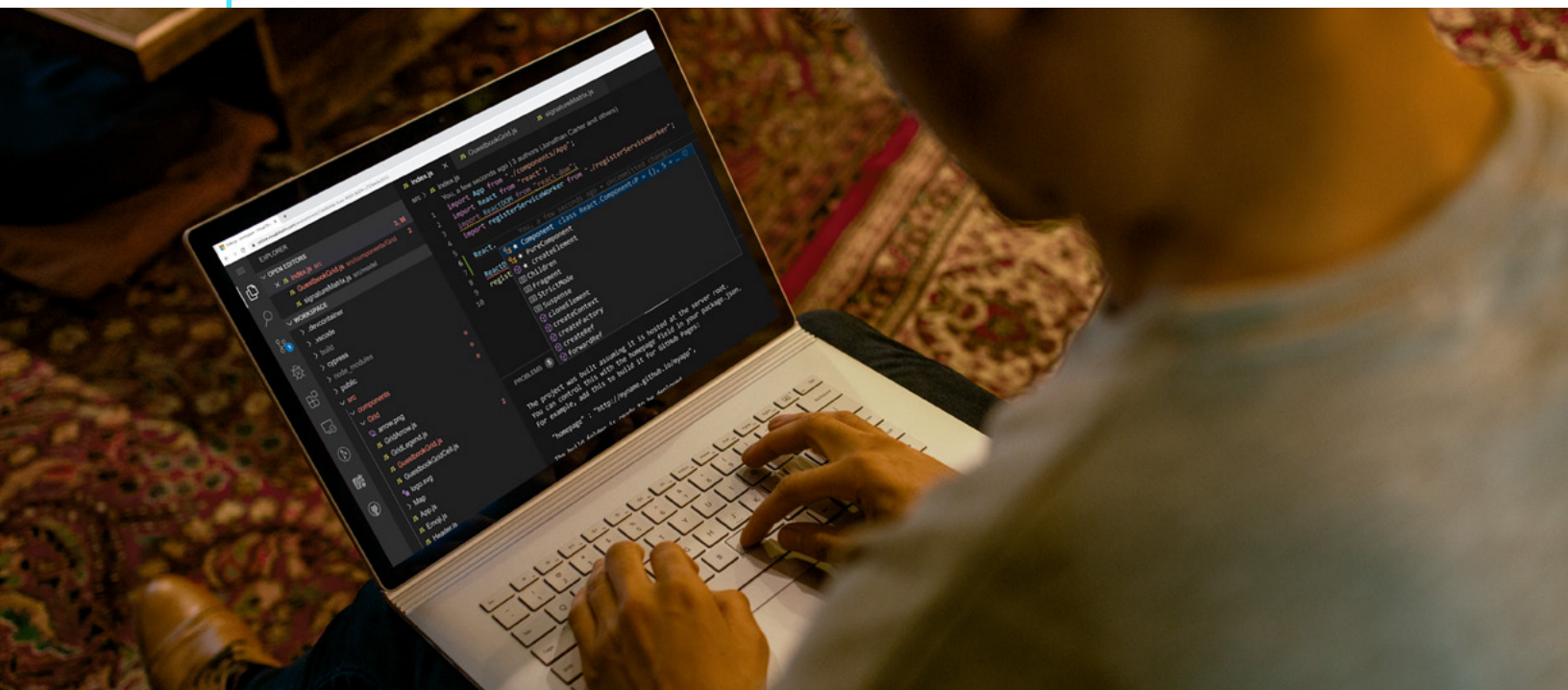
After finishing his lunchtime burrito, he hears the doorbell ring. A quick unboxing later and Ryan has his new Surface Book 3. He's excited to test it out. Luckily, IT already installed and configured his favorite host OS, so Ryan is able to jump right into the process of setting up a local development environment through his team's preconfigured VMs. Ryan knows this process will be simple from past experience, as he uses multiple local development environments on different machines for when he's traveling or working offline. He remembers his team does a good job keeping the VMs up to date so he doesn't need to manage local dependencies or manually configure tools.

While Ryan is waiting for the VM setup to complete, he gets a Microsoft Teams notification. Glancing at the screen, he sees that his earlier code change did in fact break the team build. Undeterred, Ryan fires up the web browser on his Surface Book 3 and connects to his Visual Studio Codespace, the cloud-hosted development environment his team can access from any device. "Codespaces is perfect for just this situation," he thinks.

Tip 2: Enable your team to code on-demand, from anywhere

Ryan jumps in and begins scrolling through the build logs wondering what might have gone wrong. He spots something unusual, and suspecting what could be the problem, adds a debug point. Ryan then starts the program and connects to it from another browser window. He enters some data into his running program, submits a form, and switches back to his Codespace. Execution stopped at the debug point so Ryan can inspect some variables. As suspected, he had forgotten to check for empty or null input for a particular variable before using it. He writes a unit test, makes the code change and runs the team's command line build from the console in Codespaces. Ryan is still stunned he can do this all from a web browser. Everything looks good, so he commits and pushes his changes, hoping it fixes the full build.

Ryan closes his web browser, returning to see his VM is set up. He posts in the team chat channel about the issue, noting that he's already fixed it.



Tip 3

Give your developers choice and flexibility

Once you have an established baseline of standardized environments and tools that your team can rely on, a way to optimize team performance further is to provide your developers with the flexibility and freedom to fine-tune their toolset.

This may seem counterintuitive given that standardization and continuity can be important to team productivity and collaboration—imposing a single vendor tool, for example, will ensure there aren't barriers to the team working together, speaking the same language, and following the same methodology.

But individual preferences don't need to be sacrificed. Moreover, attempting to limit developers' choices can have unintended effects, such as incentivizing people on the team to create their own mini-ecosystems of unapproved tools that can lead to security and governance vulnerabilities (also known as "shadow IT"). This risk increases when your team is fully distributed, as developers are confronted each minute with an unsupervised decision on whether to use the tool or device they prefer, or the one that's being forced on them.

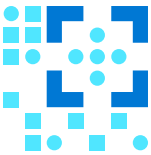
The answer? Balance your security and governance concerns with the needs of both the team and individual contributors. Have open discussions with your team, as they will likely point you to a variety of tools they already see as beneficial. You may be surprised at the flexibility you have to satisfy everyone.

Balance team productivity with individual autonomy

When determining your overall strategy, start by delineating between the areas where it's in everyone's interest to have tool continuity, relative to the areas where there's opportunity and reason to provide more flexibility. For example, it's important that your team is using the same communication and collaboration tools, including video conferencing, chat, and any important dashboards, such as a Kanban board. You'll also want to ensure that you and your remote project manager have a single source of truth spanning key project information and overall project status.

It's important that your team is using the same communication and collaboration tools.

But for development workflows, you may want to be more flexible. Those closest to a given workflow will likely have the best idea of what they need to get the job done, and that can be your baseline. From there, you can work with your team to evaluate the broader ecosystem of tooling that's either already interoperable or could be easily integrated. In some cases, it may be better in the long run to invest in developing the right ecosystem for your team rather than the approach of conformity to a single tool. Or, you may find the best of both worlds.



Choosing the right foundation

The Visual Studio family of tools is a common favorite of developer teams first and foremost because it provides a versatile and extensible platform for different workflows and environments. For example, Visual Studio Codespaces was mentioned in Tip 2 due to its ability to provide full-featured, cloud-hosted development environments, which are simple to use and accessible to distributed teams from different locations and devices. But within the Visual Studio family, teams also have the option of using other environment types, and accessing those environments via the web or a local IDE and editor that's available on Windows, Linux, or Mac devices:

- **Visual Studio: a powerful, full-featured IDE for everything from coding, debugging, testing and deployment.**
- **Visual Studio Code: an open-source code editor for editing and debugging apps on any OS. It can run locally or via a browser and features AI-assisted development spanning statement completion and refactoring.**
- **Visual Studio Live Share: a feature integrated into Visual Studio, Code, and Codespaces that's used for sharing, reviewing, and collaborating across code and projects in real time.**

Think at a systems level to identify opportunities for integration across the stack

It is important to also consider how easy it will be to add or integrate additional tools from a larger ecosystem into your workflows. Visual Studio is deeply integrated with GitHub, so your team can tap into the GitHub Marketplace and a healthy ecosystem of quality plugins that can further enhance development and automate activities (this will be covered in-depth in a subsequent tip). Visual Studio is also integrated with many Azure services, which can be immensely beneficial as your organization shifts to hybrid or public cloud environments and cloud-native applications.

Regardless of the platform and tools you adopt, the idea here is to take a sandbox approach, where you have firmly defined boundaries that maintain a strong foundation of tools and standards for consistency, collaboration, and security, yet within those boundaries your team can make individual choices on how to best achieve their goals. This flexibility and choice will also give your developer team the ability to stay on top of emerging tools and techniques as the technology landscape evolves—whether that's Docker, Kubernetes, the next great GitHub community plugin, microservices, or adopting a DevOps culture and practices.

Taken a step further, you might even want to deliberately create secure sandbox environments for your team to experiment with new ideas in a setting that is secure and has no impact to production applications. Again, with Codespaces, it's extremely easy to spin up and discard new secure environments where your team can experiment with different tools or libraries, learn a new framework, or even build a quick prototype to test out in a production-like environment.

Tip 4

Enhance remote collaboration and reduce coding friction

Ongoing collaboration and feedback are critical to keeping distributed developer teams productive.

In co-located environments, even the best developers can fall into coding silos that negatively impact productivity, particularly when code reviews are more than a few days apart. It's easy to imagine how this could get worse when each team member is remote.

In distributed environments there are fewer informal connection points for developers to share information with others on their team. Silos can grow larger, and small issues such as getting stuck on a bug or building a feature that's different from team conventions have a higher chance of spiraling into more costly and disruptive productivity inhibitors.

It's important for your developer team to remember that software is a team sport. They're trying to solve business problems quickly, but they also need to do so as a team—not as a group of individuals.

Accelerating feedback loops

To facilitate agile collaboration, start by helping to put structure in place that will accelerate feedback loops. For example, break projects up into smaller chunks and encourage people to work together in pairs. This will create additional opportunities for code reviews while putting team members in touch more frequently, helping catch issues early on for an easier and cheaper fix. This will also have the added benefit of improving code quality (and likely team morale) as developers are able to better adapt to incremental feedback and guidance.

Another way to accelerate feedback loops is by streamlining asynchronous code reviews. GitHub is a great resource for this. Its inherently distributed nature, which comes from using the distributed version control system (DVCS) Git as its foundation, is practically tailormade to support remote, asynchronous code reviews.

Your team is likely familiar with key features like pull requests in GitHub and its seamless commit process. By using these features and adopting best practices from the open source GitHub community—such as providing sufficient context in a pull request description, providing quality commentary in pull request reviews, pushing an alternative commit to articulate a different approach, and more—your team can take full advantage of its distributed nature to enhance cross-team collaboration and increase productivity. Even working across different time zones can be turned into an advantage. For example, large, globally distributed development teams have created very efficient 24-hour development pipelines through these practices and tools.

Tip 4: Enhance remote collaboration and reduce coding friction

You may have heard the term "innersourcing," which is technically what's being advocated here: fostering open, collaborative development by applying best practices from open source culture to proprietary projects and code. Innersourcing can improve the efficiency and productivity of not only your team, but the entire organization. The power of innersourcing is how it can break down the hierarchies and roadblocks that hold back the potential economies of scale that can result from widespread cross-team knowledge sharing, more rigorous peer reviews, and mass collaboration. In short, innersourcing can lead to exponentially higher quality code and productivity across large and complex software engineering organizations.

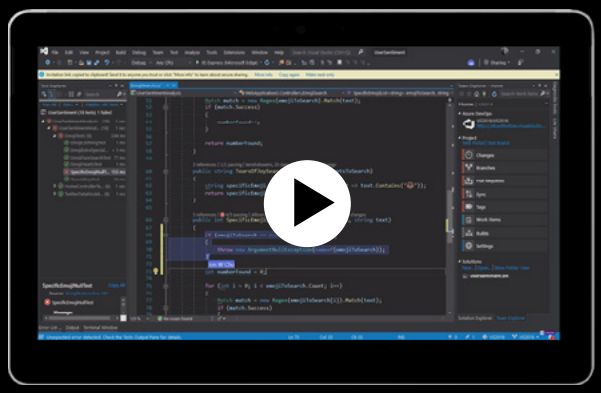
Start by helping to put structure in place that will accelerate feedback loops.

You may not be able to drive widespread adoption of innersourcing initially, but you can help set an example by showing the value of innersourcing to your organization through your team, while they get to enjoy the benefits of innersoucing across their own workflows in the meantime.

Encourage virtual pair programming

The one thing all these best practices for remote code reviews have in common is they're finding ways to replicate the efficiencies we find in a co-located environment. In some cases—such as with innersourcing and widespread asynchronous collaboration—the digital medium actually provides a benefit over physical collaboration. There's one area in particular where technology has given developers a virtual leg up over the physical world, which may come as a surprise: pair programming.

With today's latest collaboration tools, such as Visual Studio Live Share, it's entirely possible for your distributed team to take advantage of pair programming and real-time code reviews. Visual Studio Live Share enables teams to share, edit, and debug projects together with chat and audio in real time, regardless of the programming language, application being built, or even software variations such as editor versions. Visual Studio Live Share also brings unique benefits over co-located pair programming via features such as the ability for each contributor to work within their own editor preferences, track each other's work from their own individual terminal, pull helpful context, and even explore other files or ideas within the project independently while remaining plugged into the session.



Check it out

In this short video on collaboration, Allison Buchholtz-Au, a program manager for Visual Studio Codespaces, shows you helpful features like how to clone or check out code from Git repositories and how you can collaborate with anyone on the same codebase without needing to replicate their environment.

To illustrate how your distributed team can work through development roadblocks in real time with on-demand remote code reviews, let's look at the fictional story of Jay and Zoe.

Jay and Zoe's story:

Finding a remote team member to help, on demand

"I don't understand why it keeps failing! I've been working on this feature for hours," says Jay, shrugging.

Every developer knows Jay's frustrations. When working alone, missing a small detail can lead to a seemingly endless roadblock. You just can't see it yourself!

Jay makes himself a cup of tea before returning to his desk. "Maybe I should ask Zoe. She's awesome at spotting code mistakes," he thinks. "John would be great too." If their team was co-located, he could just turn around to see if either were available, but they all live in different cities. Luckily, the team is used to remote collaboration and has the right tools in place.

Jay opens the team's virtual team room—a Microsoft Teams chat channel. He can see that both Zoe and John are online and sends a quick message asking if either can help. He sees Zoe typing... "Yes!" she replies. More typing... "Send me a Live Share link and give me a call."

Jay activates Live Share from his browser-based editor in Codespaces. He messages the link to Zoe while he puts on his headset and starts an audio call. He notices Zoe has joined the Live Share workspace when her colored cursor shows up, indicating she's in the session.

Tip 4: Enhance remote collaboration and reduce coding friction

"What seems to be the problem?" Zoe asks as she inspects Jay's code from her favorite editor, Visual Studio Code. Jay responds, "I keep getting an 'undefined' error and can't figure out what's going on. It's from this block of code." He selects a block of code in his browser-based editor, knowing Live Share will show Zoe the highlighted code on her own machine.

Jay can see Zoe's cursor moving around his screen as she talks through her thoughts. "Ah, ha! I think I see the issue," she says over the call. Jay watches Zoe's cursor as it highlights a specific block of code. She continues, "In your loop here, it looks like you're using the values of the object. But you're using the JavaScript 'for/in' loop, which iterates through the properties. I think you want the 'for/of' loop instead."

"Wow!" says Jay. He knew it was right there in front of his eyes. "You're right. Thanks so much. I'll change it right away," he says before thanking her again and closing his Live Share session. "That was kind of like having Zoe coding over my shoulder," Jay thinks, as he gets back to work in his home office, complete with a fresh cup of coffee from his new espresso maker.

Tip 5

Strengthen and automate app security

Tip 3 briefly touched on the benefit of using a sandbox approach and giving your team access to the GitHub Marketplace and community to explore new tools and plugins. Who wouldn't want to tap into a huge repository of community tools and infrastructure to improve their application's capabilities and team's development workflows? But you'll want to implement proper security safeguards along the way, ensuring your team can access this goldmine of community tools while staying focused on productivity instead of spending resources on managing security concerns or hurdles.

As you'd expect with the GitHub community, there are already a number of strong tools that can scan code for security vulnerabilities. One of the most popular choices is the automated code-scanning tool Semmle. Semmle has been used for years to automatically graph and scan code when a new push request is made, checking it for common errors that can cause security vulnerabilities. Along with checking code ahead of commits, Semmle's sematic code analysis engine can also verify existing or new components across your codebase for known vulnerabilities.

What's especially powerful about Semmle is how it incorporates crowdsourced security vulnerability patterns from the entire GitHub community, enabling your team to tap into an ever-evolving breadth of security research. Semmle originally was only relied upon for open source projects, but now many enterprises are running it at scale in the cloud, enabling teams to ramp up hosted cloud environments much faster than they previously could.

Semmle's Key Capabilities:

CodeQL:

Automated variant analysis for product security

- CodeQL is a code analysis engine for product security teams to quickly find zero-days and variants of critical vulnerabilities.
- CodeQL helps you explore code quickly to find and eradicate all variants of vulnerabilities before they become a problem.
- By automating variant analysis, CodeQL enables product security teams to find zero-days and variants of critical vulnerabilities.

LGTM

Continuous security analysis for developers

- LGTM is a code analysis platform for developer teams to identify vulnerabilities early and prevent them from reaching production.
- LGTM automatically analyzes every commit to identify vulnerabilities early and enable developers to prevent zero-days from reaching production.

Regardless of the tool you use, automating app security will remove a potential headache for your team and release the enormous potential of the GitHub community, not least of which are all the ways you can automate the development lifecycle.

Tip 6

Automate, automate, automate

With a distributed team you'll want to automate as much of the development lifecycle as possible, both to remove collaboration bottlenecks and accelerate your team's ability to distribute code.

As with any flexible ecosystem, there are many popular tools you can use not only to enhance development, but also to create an end-to-end, highly automated continuous integration/continuous deployment (CI/CD) pipeline. It's worth exploring the GitHub Marketplace to find what works for your specific projects, programming languages, and workflows, but there are a few popular choices worth mentioning.

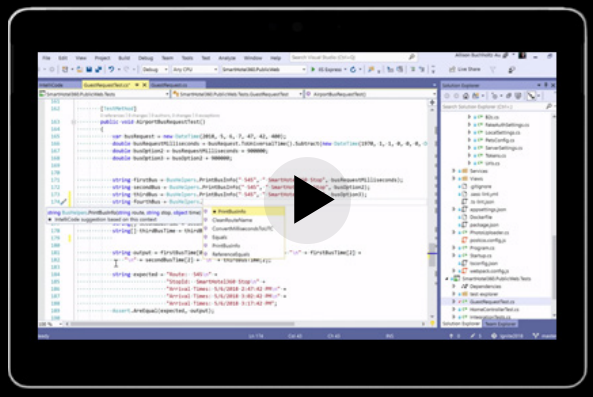
First, while the specific code editor you choose will likely be based on how it's optimized for the particular programming language you're using, there are many powerful AI-assisted editors out there that apply machine learning to writing code—making autocomplete suggestions for syntactically valid options and even anticipating what a developer will write next.

Tip 6: Automate, automate, automate

Some of these AI-assisted editors use open-source projects and popular libraries and frameworks for training, which means they can even learn more about your project over time and refine the training set.

As an example, Visual Studio Code has a capability, called IntelliCode, that makes real-time recommendations for completing statements and arguments, helps maintain coding styles and formatting conventions, facilitates automated refactoring across your code base, and shares team model recommendations by translating patterns from open-source GitHub repos to your team's code.

Another area to focus on are power tools, which can give developers more time and increase productivity by automating the basics. Examples might include picking the right data structure or algorithm or deciding on a more readable name for a variable. One such power tool is Semmle, which was highlighted in Tip 5 for its security enhancements, but is also a static code analyzer. It will debug your code by examining it against a set of coding rules, all while it's flagging security vulnerabilities.



Check it out

Learn how you can train a custom model with Microsoft Visual Studio IntelliCode to provide code suggestions based on your actual codebase and how to share your trained models, as well as how to incorporate additional language support.

Tip 6: Automate, automate, automate

One of the most useful tools you can use, due to not only its power but its flexibility, is GitHub Actions. It can be used to automate nearly any workflow throughout the development lifecycle spanning build, test, package, release, and deployment. There are thousands of plug-and-play automated actions already built and maintained by the community, such as deploying a web service, automating code reviews, automatically running static analysis, or triggering events based on a pull request. There are also many existing automations for building that CI/CD pipeline. Or your team can build custom automations based on workflow templates matching any language and tooling they want to use. GitHub Actions gives your team the ability to customize and automate their workflows, from idea to production.

GitHub Actions makes it easy to automate all your software workflows, now with world-class CI/CD.

- **Simple:** Build, test, and deploy your code right from GitHub. Make code reviews, branch management, and issue triaging work the way you want.
- **Customizable:** Run a workflow on any GitHub event. Build automations around a push, issue creation, or a new release.
- **Flexible:** Combine and configure actions for the tools and services you use, such as building a container, deploying a web service, or automate welcoming new users to a project.
- **Hosted your way:** Run on Linux, macOS, Windows, or ARM, on a VM or inside a container, and in the cloud or on-prem.
- **Supporting any language:** GitHub Actions supports Node.js, Python, Java, Ruby, PHP, Go, Rust, .NET, and more.

Tip 7

Embed signals to track app health and performance

Lastly, establishing a productive distributed team means you'll need to ensure they're able to track and respond to issues quickly throughout the development, testing, and delivery pipeline, along with properly monitoring and managing application health and performance out in the wild.

Distributed teams can't rely on a war room or physical signals to flag if an application goes down in production or there's a performance issue, so you'll need to accomplish this by building strong signals into your team's virtual workflows and environments.

Going back to the GitHub community, there are powerful tools for embedding signals across the application lifecycle and production environment. You can even look into using automation tools like GitHub Actions to trigger alerts. But in the end, to ensure your team is highly responsive, you'll likely want to invest in a centralized solution, which will populate not only key data for your dashboards, but also maintain full observability into your applications, infrastructure, and network.

Tip 7: Embed signals to track app health and performance

Let's take Azure Monitor as an example, which delivers a comprehensive solution for collecting, analyzing, and acting on data across your applications. This includes data on everything from the performance and functionality of the application code to system and resource monitoring data. It can be used to detect and diagnose application and dependency issues, drill into monitoring data for performance troubleshooting and deep diagnostics, support operations with alerts and automated actions, and even create dashboards and other visualizations for your team. Having a comprehensive solution will enable you to enhance the entire distributed development lifecycle, and most importantly, maximize the availability and performance of your applications and services.

Having a comprehensive solution will enable you to enhance the entire distributed development lifecycles.

One final point on maximizing availability, which can be an area of elevated risk for a distributed team. You'll likely have alerts set up to proactively notify your team of critical situations, but it's also important to ensure your team is set up to triage the issue quickly and effectively. Does your designated administrator or whoever is responsible for investigating an issue have immediate access to the right subject matter expert (SME)? Are there emergency protocols in place for reaching that person if they're unavailable? Are there stopgaps in place, such as standby SMEs who can support? Is critical information locked up in a silo, or have you ensured it's documented and accessible to the broader team as part of your contingency planning?

Pausing on the importance of documentation: you'll find numerous playbooks with best practices for documentation, so we won't go in-depth here, but keep in mind it's a fundamentally important concern. Like many of these tips and best practices, what might be a minor issue among a co-located team can have a disproportionate impact on a distributed team when weak points are pressure tested and the team doesn't have the necessary speed and agility to respond.

Closing thoughts

Leading a developer team is not easy. Beyond managing a set of unique individuals, each with their own talents and aspirations, you're likely working across the business to meet outcomes with good technical solutions and working hard with peers to ensure those technical solutions work well with each other. You're also likely trying to keep on top of a changing environment. Most of all, you're likely acutely aware that time is always hard to find.

It can be tempting to think you can save time by running distributed developer teams just like co-located ones. You may assume the initial challenges that naturally arise will work themselves out or can be solved later. You may see the effort required to change tools and processes as not being worth the investment of time and resources—especially if you'll need to build the case internally. But keep in mind that the earlier you start, the easier it will be to implement effective change, and the greater your return on investment will be in the end.

Many of the steps you can take to optimize the productivity of your remote developer team are simply process changes or can be implemented with the tools you already have. Where new tools or methodology shifts are required, you can pilot those changes with your team and continue in an incremental, organic way that's matched to your resources. As you and your team realize the benefits that come from improving individual and team workflows, you'll naturally secure more buy-in, and momentum will build both within your team and across the organization.

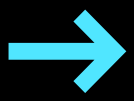
Optimizing the productivity of your remote developer team will be a unique journey. Not every recommended approach will be a direct fit, and there are no universal tools. But these tips will point you in the right direction while equipping you with the information and resources you'll need to navigate change, working through all the roadblocks and bottlenecks along the way.

Turn distributed development into an advantage. Tap into the full potential of your development talent. Get back to focusing on what matters: driving innovation and business performance with the power of code.

Microsoft

How Microsoft can help

Cutting-edge development is about building and running applications differently. To help, Microsoft offers a selection of learning resources, products, and services to help all developer teams be more productive, independent of language, framework, or cloud.



Check out the resources on the following page to get started.

Resources

Visual Studio: Try Visual Studio Codespaces, Live Share, IntelliCode, and the other tools that help your team build and collaborate more productively. Learn more.

Microsoft Azure: Turn ideas into solutions with more than 100 services to build, deploy, and manage applications—in the cloud, on-premises, and at the edge—using the tools and frameworks of your choice. Try Azure free.

Free training with Microsoft Learn: Help your developer teams gain new skills and certifications with interactive online learning to ensure developer teams become and stay proficient in their domain. Find training.

Talk to our sales team: Talk to our team of specialists to see how Microsoft can help you optimize your developer team productivity. Contact sales.

The Developer Velocity Self-Assessment Tool: Identify opportunities for your team to accelerate velocity based on a customized analysis of your current situation. Take the assessment.

Share this

Help other leaders improve their developer team productivity by sharing this series on Twitter and LinkedIn.

Share with a friend