# Summary

In this module, you set up your own private build agent by using a virtual machine that runs on Microsoft Azure.

Although a Microsoft-hosted agent often does everything you need, there are times when you might consider using your own build agent.

There are a few factors to consider when you decide to use a Microsoft-hosted agent rather than use your own. These factors include how much compute power and disk space you need and how much time is required for your builds to run.

When you configure a private build agent, it's yours to configure however you want. As a tradeoff, you also need to keep your system updated with the latest security patches and build tools.

The Tailspin web team has finished their experiment with private build agents. Let's check in to hear their results.

**Tim:** Mara and I were able to bring up a build agent running on an Azure virtual machine! The build runs quickly, and we get the same results as when we use a Microsoft-hosted agent. But does a private build agent bring any benefits to this project?

**Andy:** I agree that it was a good experiment. But for now, I don't see any benefit from using our own agent. The Microsoft-hosted agents seem to do everything we need for now, and we don't have to worry about keeping it up to date. But if we discover we need more build time, storage, or memory, it's good to know we have the option of using our own build agent.

**Tim:** I could see some of the other teams benefiting from this, especially the teams that develop the games. I know some games can require several hours to do a full build. Processing all those game assets also requires lots of CPU, memory, and disk space.

**Mara:** Good point, Tim. Because we set up that build configuration on a Git branch, those changes are isolated from the main sources. We can leave the build agent running a bit longer while you demo this to the other teams.

**Tim:** Sounds good. Thanks for your help here. This was all relatively easy to set up. And now I know little bit more about how VMs work on Azure!

## How is the team measuring up?

In the Assess your existing software development process module, Mara performed a value stream mapping exercise  to help the team analyze their current release cycle process.

Recall that the activity ratio, or efficiency, is process time divided by total lead time.

Activity ratio = Process timeTotal lead timeActivity ratio = Process timeTotal lead time

The Tailspin web team initially determined that they were 23 percent efficient against this metric.

Although the team has not yet transitioned to a full delivery cycle by using DevOps processes, they've already reduced some inefficiencies.

So far, the team has reduced:

* The time it takes to set up source control for new features from **three days** to **zero days**.

They did this by moving from a centralized source control to Git, a form of distributed source control. With distributed source control, there's no need to wait for files to be unlocked.

* The time it takes to hand off code to Amita, the tester, from **two days** to **zero days**.

They did this by moving their build process to Azure Pipelines. Azure Pipelines automatically notifies Amita when a build is available, so the developers no longer need to update Amita's spreadsheet.

* The time it takes Amita to test new features from **three days** to **one day**.

They did this by unit testing their code. Because unit tests are run each time a change moves through the build pipeline, fewer bugs and regressions reach Amita, which enables her to complete each manual test pass much faster.

These changes reduce the total lead time from 22 days to 15 days. Substitute these numbers into the equation and we get:

Activity ratio = 5 days15 days=.33Activity ratio = 5 days15 days=.33

Multiply the result by 100% and you get **33%**.

Although there's always room for improvement, this is a very positive change for the team. Not only do customers receive value more quickly, the Tailspin team now spends less time waiting and more time doing what they enjoy most: delivering features they know their customers will love.

Even management is beginning to take notice. The team plans to share their secret with management after they've had some more time to prove that the process works.

## Learning path summary

Congratulations. You've completed the final module in the Build applications with Azure DevOps learning path. In this learning path, you accomplished a lot, including:

* Setting up a project in Azure Pipelines and publishing build artifacts to the pipeline.
* Implementing a code workflow for the team members that uses Git and GitHub.
* Running automated tests, such as unit and code coverage tests, when the pipeline runs.
* Scanning for potential security vulnerabilities and related issues in both your code and the third-party components you use.
* Managing your own packages in the pipeline and connecting them to your applications.
* Using your own build agents when Microsoft-hosted agents don't meet your needs.

The focus of this learning path is on building applications and receiving build artifacts that you can hand off to your QA or operations teams.

## Continue the journey

You and the team have made a lot of progress, but the big release is coming up. How will the team deploy their build artifacts to development, test, and staging environments so they can run further test and verify their work? If you want to work along with them and learn how to configure release pipelines that continuously build, test, and deploy your applications, go to Deploy applications with Azure DevOps .

For more self-paced, hands-on learning around Azure DevOps, also check out Azure DevOps Labs .