Onboard Developers in 5 Minutes

Modern Dev Environments with Codespaces

Massimo Bonanni

Sr Technical Trainer @ Microsoft

Buy a laptop. Image it. Hope.

The traditional path from "welcome aboard" to "first commit."

PHASE 1

Procurement

Order laptop, approvals, shipping

7–14 days

PHASE 2

OS & base

OS, drivers, system updates

2–4 hrs

PHASE 3

Tooling

IDE, runtimes, CLIs, databases

1–3 hrs

PHASE 4

Repo & deps

Clone, install, fix what breaks

1–3 hrs

PHASE 5

Access

VPN, registries, tokens, secrets

1–2 hrs

TOTAL

~2 weeks calendar · 8+ hours of hands-on setup

Open a URL. Start coding.

The same journey — same five phases — through a cloud dev environment.

PHASE 1

Click

Code → Codespaces → Create

~1 sec

PHASE 2

VM ready

Dedicated VM provisioned in Azure

~15 sec

PHASE 3

Container

Image attaches (or builds)

~10 sec*

PHASE 4

Post-create

Deps install, DB seeds, scripts

30s–2 min

PHASE 5

Coding

Editor live, ports forwarded

Instant

TOTAL

~30 seconds with prebuilds · up to ~5 min cold start

Where Codespaces actually pays off

Six concrete scenarios — pick the one that's already happening in your team.

DWX

Six scenarios you've probably lived

New teammate, day one

Click here, start coding. No setup scripts, no Stack Overflow rabbit holes.

Try a new framework

Spin up Next.js, FastAPI, or Rust in 30 seconds — without polluting your laptop.

Review a pull request

Open the PR in a disposable env. Run it, poke it, throw it away.

Bisect a bug in history

Open a codespace at any commit. The bug from six months ago, reproduced today.

Code from anywhere

Tablet, Chromebook, a friend's laptop — anywhere with a browser is a workstation.

Parallel work, no clashes

One codespace per feature. No more stash → checkout → install → repeat.

What is a Codespace?

> *A development environment **hosted in the cloud** — a Docker container running on a virtual machine — that you connect to from your browser, VS Code, JetBrains, or the GitHub CLI.*
>
> ***— GitHub Docs***

Cloud-hosted

Real CPU, real RAM, real disk — not yours.

Container-based

Reproducible, isolated, defined in code.

Configuration as code

Lives in your repo. Reviewable. Versioned.

The mental model

Your repo declares the environment. GitHub builds it on demand. You get a URL.

Everyone on the team — and every future teammate — gets the same one.

From repo to running code

Three clicks. Roughly thirty seconds. No local setup required.

1

Open the repo

Code → Codespaces

2

Create codespace

Pick a config & machine

3

Start coding

VS Code in your browser

~30s

to a running editor (with prebuilds)

0

local dependencies installed

Anywhere

browser, desktop, tablet

Architecture & the lifecycle of a codespace

Editor → Internet → VM → Dev container → Your code.

DWX

How a codespace is wired together

Three layers, one TCP connection between them.

YOUR DEVICE

Browser · VS Code

JetBrains · GitHub CLI

You see code, terminal, forwarded ports.

HTTPS/ WSS

AZURE VM · LINUX HOST

Dev container

/workspaces/your-repo

Languages, runtimes, extensions, tools.

Clone

GITHUB

Repository

.devcontainer/

Source of truth for the environment.

The VM is dedicated and private to you. You have full root inside the container, limited access to the host.

What happens when you click "Create codespace"

Four steps run before you ever see a cursor blink.

01

VM + storage assigned

A dedicated VM spins up. Your repo is shallow-cloned into /workspaces.

02

Dev container is built

Docker image built from devcontainer.json (or the default image).

03

Editor connects

Browser, VS Code desktop, or `gh codespace ssh` — all hit the same container.

04

Post-create hooks run

postCreateCommand, postAttachCommand, dotfiles. Full clone finalizes in background.

With prebuilds, step 2 is already done before you click. That's how you get to ~30 seconds.

Machine types & lifecycle

Right-size the box. Stop it when you're done. Pay for what you run.

Machine spec range

Cores	RAM	Storage
2	8 GB	32 GB
4	16 GB	32 GB
8	32 GB	64 GB
16	64 GB	128 GB
32	128 GB	128 GB

Pick the smallest box that gets the job done.

Lifecycle

Running — Connected, billed by the hour for compute.

Idle timeout — 30 min default — codespace stops automatically.

Stopped — Storage only. Uncommitted changes are kept.

Deleted — Storage released. Push your branch first.

Files inside /workspaces survive stop/start AND container rebuilds.

Everything else (installed packages, ~/.bashrc tweaks, /tmp) is reset on rebuild — by design.

Dev Containers
the heart of every codespace

Customization, not personalization.
Standardize the env — keep your themes to yourself.

devcontainer.json

The contract between your repo and every future codespace.

.devcontainer/devcontainer.json

```
{
  "name": "Team Onboarding Env",
  "image": "mcr.microsoft.com/devcontainers/javascript-node:20",

  "features": {
    "ghcr.io/devcontainers/features/docker-in-docker:2": {},
    "ghcr.io/devcontainers/features/github-cli:1": {}
  },

  "forwardPorts": [3000, 5432],
  "postCreateCommand": "npm install && npm run db:seed",

  "customizations": {
    "vscode": {
      "extensions": ["dbaeumer.vscode-eslint", "esbenp.prettier-vscode"]
    }
  }
}
```

Five fields you'll actually use

image
Base container — pick a predefined or roll your own Dockerfile.

features
Plug-in installers. Docker, AWS, kubectl, languages — mix & match.

forwardPorts
Ports auto-exposed when the codespace starts.

postCreateCommand
Run once after build. Install deps, seed DBs, generate code.

customizations
VS Code settings & extensions everyone gets.

Features: building blocks for environments

Self-contained installation units that compose. Add them to devcontainer.json, rebuild, done.

- **Docker-in-Docker**
- **GitHub CLI**
- **Node.js / npm**
- **Python / pip**
- **Java / Maven**
- **kubectl + Helm**
- **AWS CLI**
- **Azure CLI**
- **Terraform**
- **PostgreSQL**
- **.NET SDK**
- **Go**

Adding a feature is three lines:

```
"features": { "ghcr.io/devcontainers/features/aws-cli:1": { "version": "latest" } }
```

Browse the catalog at containers.dev/features.

Lifecycle scripts

Hook into the boot sequence — run the right thing at the right time.

1	`onCreateCommand`	*When image is being built*	Heavy, one-off setup. Bakes into prebuilds.
2	`updateContentCommand`	*After source is updated*	Re-run dependency installs after pulls.
3	`postCreateCommand`	*Once, after container is ready*	npm install, db migrate, seed data.
4	`postStartCommand`	*Every time the codespace starts*	Start background services, daemons.
5	`postAttachCommand`	*Every time you connect an editor*	Print a welcome message, refresh tokens.

Rule of thumb: anything slow goes in onCreate so prebuilds amortize it. Anything fast goes in postCreate.

One repo, many environments

Backend dev opens a database-heavy env. Frontend opens a Playwright env. Same repo.

repository structure

```
.devcontainer/
    backend/
        devcontainer.json
        Dockerfile
    frontend/
        devcontainer.json
    ml-research/
        devcontainer.json
src/
tests/
README.md
```

Pick at creation time

Teammates see a dropdown on the codespace creation page — they choose the env that fits their task.

Configs do not inherit

Each devcontainer.json is independent. Co-locate Dockerfiles & scripts in the same subdirectory.

No config? You still get one.

GitHub ships a default image with Python, Node, Java, Go, .NET, Ruby, PHP, JupyterLab, and tooling baked in.

Power features
ports, prebuilds, complex scenarios

The things that turn a nice idea into a real platform.

Port forwarding

Run your app in the cloud — see it in your browser, share it with a teammate, ship it nowhere.

```
bash — codespace
$ npm run dev


▲ Next.js 14.0.0

- Local:    http://localhost:3000


🌐 Port 3000 forwarded to:

https://studious-octo-fortnight-3000.app.github.dev
```

Visibility

Private

Default. Only you can open the URL.

Organization

Anyone in your org with the URL.

Public

Anyone with the URL. Use sparingly.

Declare ports in devcontainer.json — your teammates get the same forwards automatically.

```
"forwardPorts": [3000, 5432]   //   "portsAttributes": { "3000": { "label": "Web app" } }
```

Prebuilds: speed at scale

For large repos, the gap between "interested" and "coding" should be measured in seconds.

WITHOUT PREBUILDS

4–8 min

to first cursor blink on a large repo

✗ Image built from scratch on every create
✗ Dependencies installed every time
✗ Full clone fetched at start

WITH PREBUILDS

~30s

on prebuild-enabled branches

✓ Image rebuilt automatically on push
✓ Deps pre-installed in the cached image
✓ Codespace attaches to ready blueprint

Configured per-branch in repo settings → Codespaces → Set up prebuild. Runs as a GitHub Actions workflow.

Four scenarios — can a Codespace do it?

Scenarios you have in mind but are afraid to ask about.

✅ *Private endpoint*

Connect to a service with no public IP.

Yes — reach it via Azure VNet integration or an in-container tunnel.

⚠️ *Mobile front-end*

Preview the app UI on a device or emulator.

Partly — real device works via the forwarded dev server; emulators stream or run locally (no KVM).

✅ *Microservices*

Run the whole stack in the codespace to test.

Yes — spin it up with Docker-in-Docker or a local kind/k3d cluster.

✅ *Local AI model*

Host a local model such as Foundry Local.

Yes — it runs in the container on CPU, served on a local port.

DEMO

Best practices for team-ready environments

Eight rules. Steal them.

Start from a predefined config
Don't write a Dockerfile until you have to. Most teams never need one.

Prefer features over RUN steps
Composable, maintained, and prebuild-friendly.

Pin versions
image:1.2.3, not :latest. Reproducibility beats novelty.

Set forwardPorts and labels
New teammates shouldn't guess which port is the API.

Don't put themes in devcontainer
Customization, not personalization. Dotfiles is the right home.

Don't bake secrets into the image
Use Codespaces Secrets and document them in the README.

Add prebuilds for big repos
If first-boot takes more than a couple of minutes, prebuild it.

Test the config in CI
devcontainers/ci builds your container on every PR.

What to remember on Monday morning

Repo == env

Your devcontainer.json IS the developer environment. PR it like code.

Onboarding ÷ 50

Half a week becomes 30 seconds — for every new teammate, forever.

Compose first

Features are LEGO bricks. Reach for them before reaching for Dockerfiles.

Prebuild big repos

If first-boot hurts, prebuilds turn the pain into a one-time CI cost.

Massimo Bonanni

Sr Technical Trainer @ Microsoft

massimo.bonanni@microsoft.com

@massimobonanni

Any Questions?

aka.ms/maxlinkedin