# 8 projects that actually get you *hired:*

# Web application with TLS certificates

Localhost is safe.

The internet is not.

Expose a real app.
Use Let's Encrypt.
Add authentication.
Add rate limiting.

Watch bots try to break in.

You need to feel what it's like when strangers can reach your services.

That's when security becomes real.

# Self-hosted Git repository with CI/CD

Build GitLab or Gitea.

Create pipelines.

Then intentionally break them.

Make tests fail.

Interviewers ask: "Tell me about a CI/CD issue you debugged."

You need a war story, not a success story.

# Kubernetes cluster running multiple applications

One app in K8s is a toy project.

Five apps talking to each other is production experience.

Deploy multiple services.

Make them communicate.
Break the networking.
Fix DNS issues.
Watch pods crash.

# Monitoring with Grafana and Prometheus

You can't troubleshoot what you can't see.

Most engineers only look at dashboards during incidents.

That's too late.
Collect metrics from everything.

Build dashboards. Watch them daily.

Learn what "normal" looks like when nothing is broken.

# Secrets management with External Secrets Operator

Secrets in code = not production-ready.

Use External Secrets Operator.
Pull from a vault.
Rotate credentials.

Never hardcode anything.

Senior engineers don't ask "where's the password?"

They ask "how do we rotate it without downtime?"

# Automated backup solution for your data

Backups you've never tested are just hope with extra steps.

Schedule backups.

Delete your database. Restore it.

Time how long it takes.

Do it again faster.

Disaster recovery is muscle memory built by repetition.

# Infrastructure as Code for your entire setup

If you can't rebuild in 30 minutes, you don't understand it.

Write Terraform for everything.

Destroy it all.

Run terraform apply.

Watch it rebuild.

Interview question: "How do you ensure consistency?"

Answer: "I destroy and recreate regularly."

# Public docs explaining your architecture

Your GitHub shows you can code.

Your documentation shows you can think.

Write how everything connects.
Draw diagrams.
Explain your choices.
Document what broke and how you fixed it.

This is proof you can explain complex systems.

That's 50% of the job.

Repost and Follow

**Mischa van den Burg**

for more content like this.