Wiz Support Technical Exercise

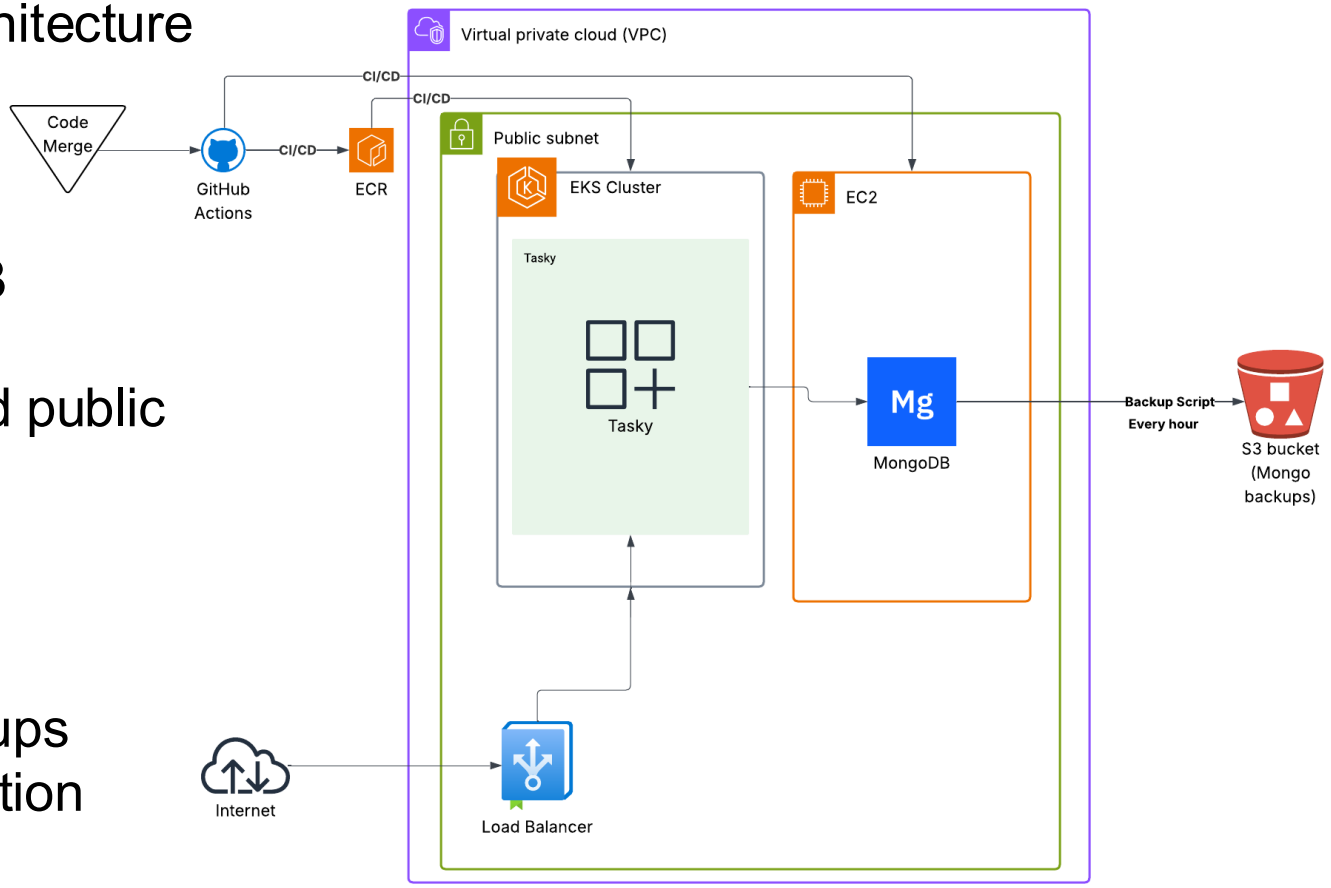
Andrew Kehr

WIZ

Overview

Designed and deployed a three-tier architecture across Kubernetes and AWS services.

- Web tier: containerized Tasky app running on EKS
- Database tier: EC2-hosted MongoDB (outdated version on outdated Linux)
- Storage tier: S3 bucket for automated public MongoDB backup



Key Requirements:

- Secure inter-tier networking
- Public accessibility for app and backups
- Database authentication with connection string
- Cluster-admin container permissions
- Backup automation with restricted DB access

What you built

Architecture Overview:

- **EKS Cluster:** Hosts containerized Go app Tasky
- **EC2 Instance:** Runs MongoDB 3.6.23 on Amazon Linux 2 (AMI ID: `ami-0c02fb55956c7d316`)
- **S3 Bucket:** `wiz-public-backups-*` with public-read access for all stored backups (https://wiz-public-backups-78d7cf71.s3.amazonaws.com/)
- **Secrets:** K8s secret used for secure DB URI injection
- **IAM:** EC2 has overly-permissive permissions for demonstration. It was given AmazonS3FullAccess along with AdministratorAccess.
- **Networking:** MongoDB (port 27017) is only accessible from within the 192.168.0.0/16 CIDR block, which includes the EKS cluster and other internal VPC resources. No public access is allowed. Port 22 is open to 0.0.0.0/0 for demonstration, though instance access is via SSM.

How you built it

- Set up using Terraform
- Created Kubernetes deployment using secret for `MONGO_URI`
- Exposed Tasky via LoadBalancer (http://af97934a7008a49f9b2397a6295e3855-672729978.us-east-1.elb.amazonaws.com/)
- Created S3 bucket with public read/list permissions
- Deployed MongoDB on EC2 manually with outdated versions and password auth and no remote root access
- Wrote `/usr/local/bin/dbbackup.sh` to run `mongodump` + `aws s3 cp`
- Automated backups using cron
- Used `wget` + busybox inside pods to confirm networking and connectivity

The challenges you faced

- **SSH Keypair Mismatch:**
  - WSL ignores chmod without an error when on a windows directory.
  - Fix was to move the .pem to the WSL ~. WSL will chmod like normal in a Linux directory
- **Used an older version of MongoDB than I was familiar with:**
  - Mongosh didn't exist yet. Mongosh was introduced in v5.0
  - Fix was to use the deprecated `Mongo` command
- **MongoDB Authentication:**
  - Typo in `Security: Authorization: Enabled` caused MongoDB to not authorize
  - Fix was obviously to fix the typo, but it was difficult to track down since the issue was completely silent and there was no warnings, errors, or logging.
- **CIDR issues:**
  - MongoDB EC2 instance initially allowed all traffic.
  - Fix was modify the security group using `authorize-security-group-ingress`
- **Connectivity Debugging:**
  - Used netstat, wget, and nc from inside busybox and EC2 to diagnose MongoDB port access

Architecture Security Outcome (Optional – Bonus)

- EC2 instance is overly privileged (intentional): full IAM access for demonstration
- Port 22 and 27017 are open to the internet
  - Scanning bots or exploit kits could take advantage of this
- S3 bucket is publicly accessible
  - Sensitive data could be downloaded by *anyone*
  - Indexable by search engines or bots
- S3 is not encrypted
- MongoDB credentials are hardcoded and also stored in plaintext
  - These could be stored as env vars
- EC2 in public subnet with public IP
  - Increases our attack surface
  - Data could be exfiltrated

Resources Used

- Terraform documentation
- AWS CLI documentation
- MongoDB 3.6 Archive Install Docs
- Kubernetes.io (Secrets, Deployments, Service Types)
- Docker documentation
- StackOverflow for SSH permission errors and netstat usage
- GitHub Actions documentation

Thank you!