1. How does cron scheduling work? Show a crontab entry to run a script every 5 minutes.

Cron is a utility on Unix-like operating systems used to schedule commands or scripts to run automatically at specified times or intervals. The entire system is driven by a background process called the "cron daemon". A user creates a "crontab" (cron table) file containing the schedule and commands, and the daemon executes them according to that plan.

How cron scheduling works

1. The cron daemon (crond) is a background process that checks crontab files and executes commands when scheduled.
2. The crontab file is where users define cron jobs, each consisting of a schedule (cron expression) and a command. You can edit your crontab using crontab -e.
3. The cron expression uses five fields for timing: minute (0-59), hour (0-23), day of month (1-31), month (1-12), and day of week (0-7, with both 0 and 7 for Sunday).
4. Special characters include * for "every" value, / for step values (e.g., */5), , for lists (e.g., 1,15,30), and - for ranges (e.g., 1-5).

Crontab entry to run a script every 5 minutes

*/5 * * * * /path/to/your/script.sh

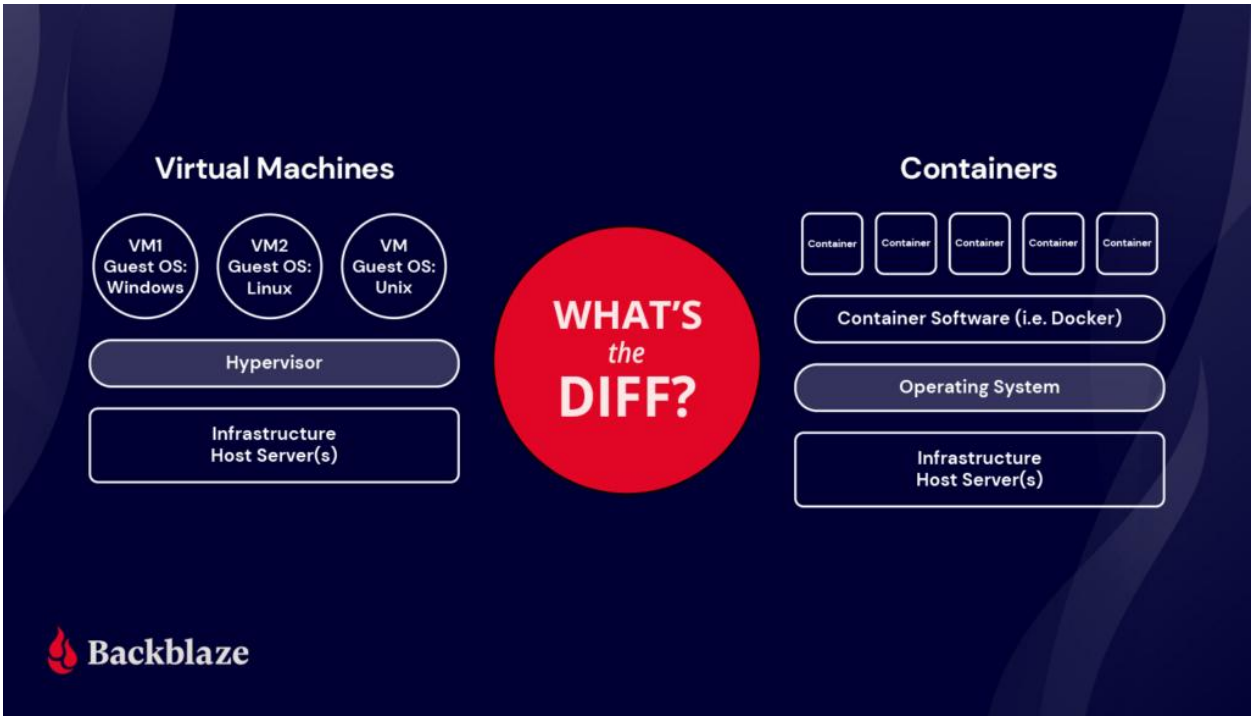
2. Why do we need log rotation? Show an example logrotate config for temperature.log.

Log rotation is an essential system administration practice for managing the growth of log files. Without it, logs can consume all available disk space, degrade system performance, and make troubleshooting difficult.

- Prevents disk space exhaustion: System and application logs can grow continuously, especially on busy servers. If left unchecked, these files can fill up the disk, leading to system instability and crashing critical services.
- Improves system performance: Working with extremely large log files can be resource-intensive. Tools like grep, tail, or log analysis platforms can slow down significantly when handling massive files, impeding effective troubleshooting.
- Aids in troubleshooting and analysis: By breaking logs into smaller, time-based or size-based files, log rotation makes it easier to find relevant information during an investigation. Instead of searching a single, massive file, you can focus on logs from a specific timeframe.
- Supports compliance: Many industries have specific data retention requirements for security and auditing purposes. Log rotation helps enforce these policies by automatically deleting the oldest logs after a defined period.
- Maintains log file organization: Rotation keeps the log directory clean and organized. Files are typically renamed with timestamps or numeric suffixes, providing a clear history of log data.

Example logrotate config for temperature.log

3. Explain the difference between a Virtual Machine and a Container. Must containers use the same OS as the host? Why or why not?



What Are VMs?

The computer-generated computers that virtualization makes possible are known as virtual machines (VMs)—separate virtual computers running on one set of hardware or a pool of hardware. Each virtual machine acts as an isolated and self-contained environment, complete with its own virtual hardware components, including CPU, memory, storage, and network interfaces. The hypervisor allocates and manages resources, ensuring each VM has its fair share and preventing interference between them.

Each VM requires its own OS. Thus each VM can host a different OS, enabling diverse software environments and applications to exist without conflict on the same machine. VMs provide a level of isolation, ensuring that failures or issues within one VM do not impact others on the same hardware. They also enable efficient testing and development environments, as developers can create VM snapshots to capture specific system states for experimentation or rollbacks. VMs also offer the ability to easily migrate or clone instances, making it convenient to scale resources or create backups.

Since the advent of affordable virtualization technology and cloud computing services, IT departments large and small have embraced VMs as a way to lower costs and increase efficiencies.

VMs, however, can take up a lot of system resources. Each VM runs not just a full copy of an OS, but a virtual copy of all the hardware that the operating system needs to run. It's why VMs are sometimes associated with the term "monolithic"—they're single, all-in-one units commonly used to run applications built as single, large files. (The nickname, "monolithic," will make a bit more sense after you learn more about containers below.) This quickly adds up to a lot of RAM and CPU cycles. They're still economical compared to running separate actual computers, but for some use cases, particularly applications, it can be overkill, which led to the development of containers.

Benefits of VMs

- All OS resources available to apps.
- Well-established functionality.
- Robust management tools.
- Well-known security tools and controls.
- The ability to run different OS on one physical machine.
- Cost savings compared to running separate, physical machines.

What Are Containers?

With containers, instead of virtualizing an entire computer like a VM, just the OS is virtualized.

Containers sit on top of a physical server and its host OS—typically Linux or Windows. Each container shares the host OS kernel and, usually, the binaries and libraries, too, resulting in more efficient resource utilization. (See below for definitions if you're not familiar with these terms.) Shared components are read-only.

Why are they more efficient? Sharing OS resources, such as libraries, significantly reduces the need to reproduce the operating system code—a server can run multiple workloads with a single operating

system installation. That makes containers lightweight and portable—they are only megabytes in size and take just seconds to start. What this means in practice is you can put two to three times as many applications on a single server with containers than you can with a VM. Compared to containers, VMs take minutes to run and are an order of magnitude larger than an equivalent container, measured in gigabytes versus megabytes.

Container technology has existed for a long time, but the launch of Docker in 2013 made containers essentially industry standard for application and software development. Technologies like Docker or Kubernetes to create isolated environments for applications. And containers solve the problem of environment inconsistency—the old "works on my machine" problem often encountered in software development and deployment.

Developers generally write code locally, say on their laptop, then deploy that code on a server. Any differences between those environments—software versions, permissions, database access, etc.—leads to bugs. With containers, developers can create a portable, packaged unit that contains all of the dependencies needed for that unit to run in any environment whether it's local, development, testing, or production. This portability is one of containers' key advantages.

Containers also offer scalability, as multiple instances of a containerized application can be deployed and managed in parallel, allowing for efficient resource allocation and responsiveness to changing demand.

Microservices architectures for application development evolved out of this container boom. With containers, applications could be broken down into their smallest component parts or "services" that serve a single purpose, and those services could be developed and deployed independently of each other instead of in one monolithic unit.

For example, let's say you have an app that allows customers to buy anything in the world. You might have a search bar, a shopping cart, a

buy button, etc. Each of those “services” can exist in their own container, so that if, say, the search bar fails due to high load, it doesn’t bring the whole thing down. And that’s how you get your Prime Day deals today.

Benefits of Containers

- Reduced IT management resources.
- Faster spin ups.
- Smaller size means one physical machine can host many containers.
- Reduced and simplified security updates.
- Less code to transfer, migrate, and upload workloads.

What’s the Diff: VMs vs. Containers

The virtual machine versus container debate gets at the heart of the debate between traditional IT architecture and contemporary DevOps practices.

VMs have been, and continue to be, tremendously popular and useful, but sadly for them, they now carry the term “monolithic” with them wherever they go like a 25-ton Stonehenge around the neck. Containers, meanwhile, pushed the old gods aside, bedecked in the glittering mantle of “microservices.” Cute.

To offer another quirky tech metaphor, VMs are to containers what glamping is to ultralight backpacking. Both equip you with everything you need to survive in the wilds of virtualization. Both are portable, but containers will get you farther, faster, if that’s your goal. And while VMs bring everything and the kitchen sink, containers leave the toothbrush at home to cut weight. To make a more direct comparison, we’ve consolidated the differences into a handy table:

VMs	Containers
Heavyweight.	Lightweight.
Limited performance.	Native performance.
Each VM runs in its own OS.	All containers share the host OS.
Hardware-level virtualization.	OS virtualization.
Startup time in minutes.	Startup time in milliseconds.
Allocates required memory.	Requires less memory space.
Fully isolated and hence more secure.	Process-level isolation, possibly less secure.

Uses for VMs vs. Uses for Containers

Both containers and VMs have benefits and drawbacks, and the ultimate decision will depend on your specific needs.

When it comes to selecting the appropriate technology for your workloads, virtual machines (VMs) excel in situations where applications demand complete access to the operating system's resources and functionality. When you need to run multiple applications on servers, or have a wide variety of operating systems to manage, VMs are your best choice. If you have an existing monolithic application that you don't plan to or need to refactor into microservices, VMs will continue to serve your use case well.

Containers are a better choice when your biggest priority is maximizing the number of applications or services running on a minimal number of servers

and when you need maximum portability. If you are developing a new app and you want to use a microservices architecture for scalability and portability, containers are the way to go. Containers shine when it comes to cloud-native application development based on a microservices architecture.

You can also run containers on a virtual machine, making the question less of an either/or and more of an exercise in understanding which technology makes the most sense for your workloads.

Must containers use the same OS as the host? Why or why not?

Yes, containers must use the same operating system kernel as the host. The core principle of containerization is OS-level virtualization, not hardware virtualization. This means that the container engine (like Docker) simply uses features of the host's kernel to create an isolated user-space for each container.
Because the container shares the host's kernel:

- A Linux container cannot natively run on a Windows host.
- A Windows container cannot natively run on a Linux host.

However, there are workarounds, especially on desktops, which can create a misconception that they are compatible:

- Docker Desktop on Windows/macOS: When you run a Linux container on a Windows or macOS machine, Docker doesn't magically run the Linux kernel. Instead, it spins up a lightweight Linux VM (e.g., via WSL2 on Windows or a Hypervisor on macOS) to act as the host for your Linux containers.
- Hyper-V Isolation on Windows: Windows allows for containers with Hyper-V isolation. This effectively creates a tiny, highly-optimized VM for each container, which gives it a separate kernel and allows for running different versions of the Windows OS. This is different from the standard process-level isolation.

4. Reflection: Which actions in this project combined multiple Linux concepts (e.g., redirection + process monitoring)? How does this apply to real IoT systems?

log and log rotation Actions

A real-world IoT system project that uses a "log action" to retrieve data every 5 minutes is an environmental monitoring system for a greenhouse.
We can make a project to maintain optimal growing conditions within a greenhouse by continuously monitoring key environmental parameters and logging the data for analysis and automated adjustments.

Data Retrieval and Logging:

- The microcontroller is programmed to read data from all connected sensors every 5 minutes.
- This data is then formatted and transmitted to a central IoT platform (e.g., AWS IoT, Google Cloud IoT Core, or a self-hosted platform like Node-RED with a database).
- The "log action" refers to the process where the IoT platform receives this data and stores it in a time-series database (e.g., InfluxDB, PostgreSQL with TimescaleDB extension). Each entry in the log includes the timestamp, sensor ID, and the measured value.

Data Utilization:

- **Real-time Monitoring:** A dashboard displays the current environmental conditions within the greenhouse.
- **Historical Analysis:** The logged data allows for trend analysis, identifying patterns, and understanding the impact of environmental fluctuations on plant growth.

- **Alerting:** Rules can be set up to trigger alerts (e.g., email, SMS) if any parameter falls outside predefined thresholds (e.g., temperature too high, soil moisture too low).
- **Automation:** The system can also be integrated with actuators to automatically control greenhouse equipment, such as turning on fans for cooling, activating irrigation systems, or adjusting grow lights based on the logged data and predefined rules.