Monitoring the uptime of a website with Bash, cron and systemd

Ensuring that a website remains online and accessible is crucial for any business or individual relying on web presence. Automated monitoring can help quickly identify downtime and performance issues, allowing for swift corrective actions.

This Bash script checks the status of a website at regular intervals and logs whether the site is up or down along with the HTTP status code and the time taken for each check.

```
#!/bin/bash

# Base URL of the site to monitor
site_url=$1

# Delay between requests in seconds
delay_between_requests=60

# Function to check site status
check_site() {
    # Define the user agent for the request
    local user_agent="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"

    # Record the start time of the request
    local start_time=$(date +%s)

    # Get the HTTP status code from the request
    local status_code=$(curl -H "$user_agent" -s -o
/dev/null -w "%{http_code}" $site_url)

    # Get the current date and time for logging
    local date_time_log=$(date +"%Y-%m-%d %H:%M:%S")

    # Record the end time of the request
    local end_time=$(date +%s)

    # Calculate the elapsed time for the request
    local elapsed_time=$((end_time - start_time))

    # Log the status based on the HTTP status code
    if [ $status_code -ne 200 ]; then
        echo "[$date_time_log]: The site is down.
Status code: $status_code - Elapsed time:
$elapsed_time seconds"
    else
        echo "[$date_time_log]: The site is up.
Status code: $status_code - Elapsed time:
$elapsed_time seconds"
    fi
}

# Infinite loop to repeatedly check the site
while true; do
    check_site
    sleep $delay_between_requests
done
```

How the script works:

1. **URL Input**: The script takes the website URL as an argument when executed.

2. **Request Interval**: The variable `delay_between_requests` specifies the delay in seconds between each check, set to 60 seconds by default.

3. **User Agent**: The `user_agent` variable is set to mimic a request from a common web browser, ensuring that the request is handled similarly to a real user visit.

4. **HTTP Request and Timing**: The `check_site` function:

    - Records the start time.
    - Sends an HTTP GET request using `curl`, capturing the HTTP status code.
    - Records the end time.
    - Calculates the elapsed time for the request.

5. **Logging**: The script logs the current date and time, the HTTP status code, and the elapsed time for each check. If the status code is not 200, it indicates that the site is down.

6. **Infinite Loop**: The script runs indefinitely, checking the site's status every 60 seconds (or the specified interval).

Using cron

To execute the script via `cron`, removing the infinite loop is necessary because `cron` will handle the periodic execution of the script. Here's how you can modify the script and set it up with `cron`.

Remove the infinite loop and the `sleep` command from the script:

```
#!/bin/bash
```

```
# Base URL of the site to monitor
site_url=$1

# Function to check site status
check_site() {
    # Define the user agent for the request
    local user_agent="Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/58.0.3029.110 Safari/537.36"

    # Record the start time of the request
    local start_time=$(date +%s)

    # Get the HTTP status code from the request
    local status_code=$(curl -H "$user_agent" -s -o
/dev/null -w "%{http_code}" $site_url)

    # Get the current date and time for logging
    local date_time_log=$(date +"%Y-%m-%d %H:%M:%S")

    # Record the end time of the request
    local end_time=$(date +%s)

    # Calculate the elapsed time for the request
    local elapsed_time=$((end_time - start_time))

    # Log the status based on the HTTP status code
    if [ $status_code -ne 200 ]; then
        echo "[$date_time_log]: The site is down.
Status code: $status_code - Elapsed time:
$elapsed_time seconds"
    else
        echo "[$date_time_log]: The site is up.
Status code: $status_code - Elapsed time:
$elapsed_time seconds"
    fi
}

# Call the check_site function
check_site
```

Save the modified script to a file, for example, `check_site.sh` and ensure the script has execute permissions.

```
chmod a+x check_site.sh
```

Now open the crontab editor for the current user and add a new line to the crontab to schedule the script. For example, to run the script every 5 minutes:

```
*/5 * * * * /path/to/check_site.sh
https://example.com >> /path/to/logfile.log 2>&1
```

This line does the following:

- `*/5 * * * *`: Runs the script every 5 minutes.
- `/path/to/check_site.sh https://example.com`: Specifies the path to the script and the URL to check.
- `>> /path/to/logfile.log 2>&1`: Appends the output to `logfile.log` and redirects errors to the same file.

By setting up the script with `cron`, you offload the periodic execution responsibility to the cron daemon, ensuring the script runs at your specified intervals without the need for an infinite loop within the script itself.

Using systemd

To execute the script via `systemd`, you'll need to create a systemd service unit file and a timer unit file to schedule the script execution.

Remove the infinite loop and the `sleep` command from the script:

```
#!/bin/bash

# Base URL of the site to monitor
site_url=$1

# Function to check site status
check_site() {
    # Define the user agent for the request
    local user_agent="Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/58.0.3029.110 Safari/537.36"

    # Record the start time of the request
    local start_time=$(date +%s)

    # Get the HTTP status code from the request
    local status_code=$(curl -H "$user_agent" -s -o
/dev/null -w "%{http_code}" $site_url)

    # Get the current date and time for logging
    local date_time_log=$(date +"%Y-%m-%d %H:%M:%S")
    # Record the end time of the request
    local end_time=$(date +%s)

    # Calculate the elapsed time for the request
    local elapsed_time=$((end_time - start_time))

    # Log the status based on the HTTP status code
    if [ $status_code -ne 200 ]; then
        echo "[$date_time_log]: The site is down.
Status code: $status_code - Elapsed time:
$elapsed_time seconds"
    else
        echo "[$date_time_log]: The site is up.
Status code: $status_code - Elapsed time:
$elapsed_time seconds"
    fi
}

# Call the check_site function
check_site
```

Save the modified script to a file and make sure the script has execute permissions. Then create a new file at `/etc/systemd/system/check_site.service` with the following content:

```
[Unit]
Description=Check Site Status

[Service]
Type=oneshot
```

```
ExecStart=/usr/local/bin/check_site.sh
https://example.com
```

Now create the systemd timer unit file by adding a new file at `/etc/systemd/system/check_site.timer` with the following content:

```
[Unit]
Description=Run Check Site Script Every 5 Minutes

[Timer]
OnBootSec=5min
OnUnitActiveSec=5min
Unit=check_site.service

[Install]
WantedBy=timers.target
```

Reload the systemd manager configuration to recognize the new unit files.

```
sudo systemctl daemon-reload
```

Enable the timer to start at boot.

```
sudo systemctl enable check_site.timer
```

Start the timer immediately.

```
sudo systemctl start check_site.timer
```

Finally, verify that the timer is active.

```
sudo systemctl status check_site.timer
```

By default, the output of the script run by `systemd` will be captured by the `journalctl` logs. You can view the logs using:

```
journalctl -u check_site.service
```

If you want to log the output to a specific file, you can modify the service file:

```
[Service]
Type=oneshot
ExecStart=/usr/local/bin/check_site.sh
https://example.com
StandardOutput=append:/var/log/check_site.log
StandardError=append:/var/log/check_site.log
```

By setting up the script with `systemd`, you delegate the responsibility of periodic execution to `systemd`'s timer units. This approach is robust and integrates well with the system's init process, providing better management and logging capabilities compared to `cron`.

Conclusion

This simple Bash script provides a foundational tool for monitoring website uptime. By adjusting its parameters and expanding its functionalities, it can be tailored to meet more complex monitoring needs. Automating such tasks ensures that website administrators can stay informed about their site's status and respond quickly to any issues that arise.