

# Export Database with Kubernetes Cronjob

- [s3cmd Configuration](#)
- [Build Cronjob](#)
- [Security Warning](#)
- [s3cmd Upload to S3](#)
- [s3cmd Exclude Folder When Upload](#)

# s3cmd Configuration

## Example of `~/.s3cfg` Configuration

```
[default]
access_key = YOUR_ACCESS_KEY
secret_key = YOUR_SECRET_KEY
host_base = s3.amazonaws.com
host_bucket = %(bucket)s.s3.amazonaws.com
use_https = True
region = us-east-1
signature_v2 = False
encrypt = False
```

Read detail here if you are working on Digital Ocean:

- <https://docs.digitalocean.com/products/spaces/reference/s3cmd/>

# Build Cronjob

## Bash Script to Export and Upload DB Backup

```
#!/bin/bash

# MySQL Backup and S3 Upload Script with s3cmd
# Usage: ./mysql_backup_and_upload.sh

# Configuration Variables (can be replaced by environment variables)
MYSQL_HOST="${MYSQL_HOST:-mysql-service}"
MYSQL_USER="${MYSQL_USER:-root}"
MYSQL_PASSWORD="${MYSQL_PASSWORD:-your-mysql-password}"
BACKUP_DIR="${BACKUP_DIR:-/mnt/introvesia_pvc01/dbs/backups}"
S3_BUCKET="${S3_BUCKET:-your-s3-bucket}"
RETENTION_DAYS="${RETENTION_DAYS:-7}"

# Logging function
log() {
    echo "[$(date +%Y-%m-%d %H:%M:%S)] $*"
}

# Ensure backup directory exists
mkdir -p "${BACKUP_DIR}"

# Generate timestamp for backup file
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="${BACKUP_DIR}/mysql_backup_${TIMESTAMP}.sql"
COMPRESSED_BACKUP_FILE="${BACKUP_FILE}.gz"

# Create s3cmd configuration file
S3CMD_CONFIG="/tmp/s3cfg"
```

```
cat << EOF > "${S3CMD_CONFIG}"
[default]
access_key = ${AWS_ACCESS_KEY_ID}
secret_key = ${AWS_SECRET_ACCESS_KEY}
host_base = s3.amazonaws.com
host_bucket = %(bucket)s.s3.amazonaws.com
use_https = True
EOF

# Perform MySQL Backup
log "Starting MySQL backup..."
if ! mysqldump -h "${MYSQL_HOST}" -u "${MYSQL_USER}" -p"${MYSQL_PASSWORD}" --all-databases >
"${BACKUP_FILE}"; then
    log "ERROR: MySQL backup failed"
    exit 1
fi

# Compress the backup
log "Compressing backup file..."
if ! gzip "${BACKUP_FILE}"; then
    log "ERROR: Backup compression failed"
    exit 1
fi

# S3 Upload using s3cmd
log "Uploading backup to S3..."
# Create S3 path with daily subdirectory
S3_PATH="s3://${S3_BUCKET}/mysql-backups/$(date +"%Y-%m-%d")/"

# Upload to S3
if ! s3cmd -c "${S3CMD_CONFIG}" put "${COMPRESSED_BACKUP_FILE}" "${S3_PATH}"; then
    log "ERROR: S3 upload failed"
    exit 1
fi

# Cleanup: Remove local backups older than RETENTION_DAYS
log "Cleaning up old local backups..."
find "${BACKUP_DIR}" -name "*.sql.gz" -type f -mtime +${RETENTION_DAYS} -delete
```

```
# Log success
log "Backup and upload completed successfully"

# Remove temporary s3cmd config
rm "${S3CMD_CONFIG}"
```

# Create Cronjob

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-backup-credentials
  namespace: db
type: Opaque
stringData:
  MYSQL_HOST: mysql-service
  MYSQL_USER: root
  MYSQL_PASSWORD: your-secure-password
  BACKUP_DIR: /mnt/introvesia_pvc01/dbs/backups
  S3_BUCKET: your-s3-bucket-name
  AWS_ACCESS_KEY_ID: your-aws-access-key
  AWS_SECRET_ACCESS_KEY: your-aws-secret-key
```

---

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: mysql-backup-and-upload
  namespace: db
spec:
  schedule: "0 2 * * *" # Run daily at 2 AM
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: OnFailure
          containers:
```

```
- name: mysql-backup
image: mysql:8.0
command:
- /bin/sh
- -c
- |
# Install dependencies
apt-get update && apt-get install -y wget python3-pip

# Install s3cmd
pip3 install s3cmd

# Copy backup script to container
cp /scripts/mysql_backup_and_upload.sh /tmp/mysql_backup_and_upload.sh
chmod +x /tmp/mysql_backup_and_upload.sh

# Run backup script
/tmp/mysql_backup_and_upload.sh
env:
- name: MYSQL_HOST
valueFrom:
secretKeyRef:
name: mysql-backup-credentials
key: MYSQL_HOST
- name: MYSQL_USER
valueFrom:
secretKeyRef:
name: mysql-backup-credentials
key: MYSQL_USER
- name: MYSQL_PASSWORD
valueFrom:
secretKeyRef:
name: mysql-backup-credentials
key: MYSQL_PASSWORD
- name: BACKUP_DIR
valueFrom:
secretKeyRef:
name: mysql-backup-credentials
key: BACKUP_DIR
- name: S3_BUCKET
```

```
valueFrom:
  secretKeyRef:
    name: mysql-backup-credentials
    key: S3_BUCKET
- name: AWS_ACCESS_KEY_ID
  valueFrom:
    secretKeyRef:
      name: mysql-backup-credentials
      key: AWS_ACCESS_KEY_ID
- name: AWS_SECRET_ACCESS_KEY
  valueFrom:
    secretKeyRef:
      name: mysql-backup-credentials
      key: AWS_SECRET_ACCESS_KEY
volumeMounts:
- name: backup-storage
  mountPath: /mnt/introvesia_pvc01/dbs/backups
- name: backup-script
  mountPath: /scripts
volumes:
- name: backup-storage
  hostPath:
    path: /mnt/introvesia_pvc01/dbs/backups
    type: Directory
- name: backup-script
  configMap:
    name: mysql-backup-script
    defaultMode: 0755
```

---

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-backup-script
  namespace: db
data:
  mysql_backup_and_upload.sh: |
    #!/bin/bash
```

```
# MySQL Backup and S3 Upload Script
# (Same script as in the previous artifact)
# ... (paste the entire script from the previous artifact here)
```



# Security Warning

WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /root/.kube/config

It's crucial to secure your Kubernetes configuration files to prevent unauthorized access. Here are a few steps you can take to improve security:

1. **File Permissions:** Ensure that only the necessary user or group has read and write permissions on the configuration file. You can change permissions using `chmod`:

```
chmod 600 /root/.kube/config
```

This command restricts read and write permissions to the file owner only.

2. **Use Environment Variables:** Instead of relying on a configuration file, you can use Kubernetes environment variables like `KUBECONFIG` to specify the configuration file path. This allows you to keep sensitive information out of files and set permissions accordingly.
3. **Minimize Access:** Limit access to the configuration file to only those users or processes that absolutely need it. Avoid sharing this file unnecessarily or storing it in locations accessible by multiple users.
4. **Encryption:** Consider encrypting sensitive data within the configuration file. Tools like `kubeseal` can encrypt specific fields or entire configuration files for added security.

By implementing these practices, you can significantly reduce the risk of unauthorized access to your Kubernetes cluster configuration.

# s3cmd Upload to S3

To upload folders to an Amazon S3 bucket using `s3cmd`, you can follow these steps. `s3cmd` is a command-line tool for managing Amazon S3 and other cloud storage services. Make sure you have `s3cmd` installed and configured with your AWS credentials before proceeding.

## Step-by-Step Guide

### 1. Install `s3cmd`

If `s3cmd` is not already installed on your system, you can typically install it using package managers like `apt` (for Debian/Ubuntu) or `brew` (for macOS). Here are some example commands:

- **Debian/Ubuntu:**

```
sudo apt-get update
sudo apt-get install s3cmd
```

- **macOS** (using Homebrew):

```
brew install s3cmd
```

Make sure to configure `s3cmd` with your AWS credentials after installation:

```
s3cmd --configure
```

Follow the prompts to enter your AWS Access Key ID, Secret Access Key, default region, and other configuration options.

### 2. Upload a Folder to S3

To upload a folder and its contents to an S3 bucket using `s3cmd`, use the `sync` command. Here's the basic syntax:

```
s3cmd sync /path/to/local/folder s3://your-bucket-name/path/in/s3
```

- `/path/to/local/folder`: Replace this with the path to the local folder you want to upload.
- `s3://your-bucket-name/path/in/s3`: Replace `your-bucket-name` with your actual S3 bucket name and specify the desired path within the bucket.

For example, to upload a local folder named `myfolder` to an S3 bucket named `my-bucket`:

```
s3cmd sync myfolder s3://my-bucket/
```

This command recursively uploads all files and subdirectories within `myfolder` to the root of `my-bucket` in S3.

### 3. Additional Options

- **Preserve Metadata:** Use `--preserve` to preserve file metadata during synchronization.
- **Delete Removed Files:** Add `--delete-removed` to delete objects in S3 that are not present locally.
- **Configure ACLs:** Use `--acl-public` or `--acl-private` to set access control lists (ACLs) for uploaded files.

Refer to the `s3cmd` documentation (`man s3cmd` or `s3cmd --help`) for more options and customization.

## Notes:

- **AWS Credentials:** Ensure your AWS credentials (`AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`) are correctly configured for `s3cmd`.
- **Permissions:** Make sure the AWS credentials have sufficient permissions to upload files to the specified S3 bucket.
- **Security:** Handle AWS credentials securely and consider using IAM roles with appropriate permissions instead of long-term access keys.

Using `s3cmd`, you can efficiently upload entire folders and their contents to Amazon S3, making it a convenient tool for managing cloud storage from the command line. Adjust commands and options based on your specific requirements and environment.

# s3cmd Exclude Folder When Upload

When using `s3cmd` to sync or upload files to an S3 bucket, you can exclude specific files or folders using the `--exclude` option. This allows you to selectively upload content while excluding certain directories or files from being transferred.

## Syntax for Excluding a Folder

To exclude a folder (or multiple folders) during the synchronization or upload process with `s3cmd`, follow this syntax:

```
s3cmd sync /path/to/local/folder s3://your-bucket-name/path/in/s3 --exclude "folder_name/*"
```

- `/path/to/local/folder`: Path to the local folder you want to sync/upload.
- `s3://your-bucket-name/path/in/s3`: S3 bucket destination path.
- `--exclude "folder_name/*"`: Specifies the folder name or pattern to exclude. Replace `folder_name` with the name of the folder you want to exclude.

## Example

For instance, if you have a local folder named `myfolder` and you want to exclude a subfolder named `excluded_folder` when uploading to an S3 bucket named `my-bucket`, you can use:

```
s3cmd sync myfolder s3://my-bucket/ --exclude "excluded_folder/*"
```

This command ensures that the contents of `excluded_folder` within `myfolder` are not uploaded to `my-bucket`.

## Additional Notes

- **Multiple Exclusions:** You can specify multiple `--exclude` options to exclude multiple folders or patterns. For example:

```
s3cmd sync myfolder s3://my-bucket/ --exclude "excluded_folder/*" --exclude "other_folder/*.txt"
```

- **Using Patterns:** The `--exclude` option supports glob patterns (`*`, `?`, `[...]`) to match multiple files or folders based on a pattern.
- **Preserve Metadata:** Optionally use `--preserve` to preserve file metadata during synchronization, and `--delete-removed` to delete objects in S3 that are not present locally.
- **Consult Documentation:** For more advanced usage or specific cases, refer to the `s3cmd` documentation (`man s3cmd` or `s3cmd --help`) for comprehensive details on its options and capabilities.

By leveraging the `--exclude` option in `s3cmd`, you can tailor your uploads to S3 to exclude specific folders or files as needed, ensuring flexibility and control over your synchronization operations. Adjust commands and options based on your specific requirements and environment.