Snapshots in EVE

Currently, support for snapshot functionality will only be available in EVE in a ZFS storage configuration.

Requirements and motivation:

After we began to actively develop the ZFS storage in EVE, our users had a requirement in functionality for working with snapshots, namely, at the moment we want to have support for the following functionality:

1. Should be able to provide an ability to snapshot a storage volume and store the snapshot locally;
2. Should be able to create multiple snapshots of each volume;
3. Should be able to rollback any volume to a given snapshot (without causing data corruption);

EVE doesn't need to transfer the snapshot out of the local storage on the other device.
All external backup and restores should be done as part of application data backup by a 3rd party or by cluster storage.

Snapshots are not backups

It is important to clarify, that while snapshots are providing time machine capabilities, it is not a backup alternative. Similarly, a raid is not a backup. If the disk is broken, or the whole node gets in the fire/flood/tornado/etc, all the data, including every snapshot is gone.

Furthermore, an application (e.g. a database), knows better how to backup itself more efficiently. Such backup usually would take much less disk space, as it does not contain the application itself. And generally uploading these backups to a cloud (e.g. s3) is a better disaster recovery strategy.

Snapshots, on the other hand, are a perfect first-line defense. If something is going wrong in the application, it is very easy to rollback the whole system to an earlier state.

To summarize, we want to be communicated to users that **snapshots do not qualify as a disaster recovery strategy**. But they are a good addition to an existing strategy.

Possible user stories

Cases that do not require consistency with applications:

1. Creating a snapshot immediately after creating a logical volume and before the first launch of the application. This will allow us to reset the virtual machine to its initial state without recreating the logical volume (We are now recreating the logical volumes). Basically, it just simplifies the current process in EVE to clear the logical volume for the VM without adding anything new.
2. Creating/rollback a snapshot when the application is turned off. This is considered as a basic capability that does not need to be consistent with applications. Can be used as a checkpoint that does not imply conflicts with rollback when the VM is powered off. User story for this case can be anything, such as rollback a logical volume to state N after an unsuccessful update.

Cases where application consistency is needed (Main query):

As administrator:

1. I need to be able to create a snapshot of the logical volume where stored database N and used by application N, for example before important updates once a week.
2. I need to be able to rollback to snapshot N if something went wrong and, for example, data was corrupted in the database after the actions of user X.
3. I would like to be able to get information about snapshots on the controller, for example, to understand the space occupied by snapshots and their status.
4. I need to be able to manage snapshots through the controller (create/delete/rollback). For example, EVE has run out of space or is not enough to create a new VM. Thus, as an administrator, I can delete the old snapshot via controller that I no longer need, thereby freeing up space for the new VM.

As a VM usual user without controller access:

1. After I manually or through a script paused the I/O of application N and reset its cache (if the application has such functionality), I need to be able to send a command to create or rollback a snapshot, and in case of a positive or negative outcome, receive information about this event from EVE. (for example, lack of space when creating a snapshot, a successful rollback operation, and other information or problems)
2. Be have able to get/view a list of available snapshots for a specific disk, available for rollback or deletion on the VM side.

Crash-Consistent snapshots vs application-consistent snapshots

The Crash-Consistent snapshot contains an immediate state of the disk. As if there was a power cut-off at the point of snapshot creation. Whatever was in memory is lost. For robust applications, Crash-Consistent snapshots are enough because they are designed to tolerate power cut-offs.

The application-consistent snapshot is created with the collaboration of the app running in the VM. Of course, the application needs to be aware of the snapshot being taken to flush all the data before it happens.

Application concern of EVE-OS volume snapshots

Problem statement

The main target is to make EVE-OS take ZFS snapshots of the volumes of an application instance when the controller requests (using some new EVE API) and also add an EVE API to request that the application instance restart from a particular snapshot. This is very efficient and the ZFS snapshots are atomic.

Problem

However, the concern is around applications and guest VMs which have their own internal buffering and/or ordering of writes to their virtual disks. Even though the ZFS snapshots are perfectly consistent such an application instance might never be able to successfully run using a snapshot.

Typically a guest VM (kernel) will not have much of an issue. It might have some kernel buffers which are not written in which case some fsck might run when booting guest VM from the snapshot.

But if there is some (home-grown or otherwise) database-like application running in the guest VM then all bets are off since we do not know what assumptions it makes. We know that such applications exist because we've been requested to add support for the graceful shutdown of applications since they can not handle the box just being powered off.

Thus if we promise that the snapshots are usable we might end up with very unhappy users.

Possible Solution

At a minimum, we need to make it clear that the custom application needs to make sure it can be restarted from an arbitrary snapshot, but we don't have much confidence that will be sufficient. If there was an API where EVE-OS would tell the app instance "flush all of your application buffers now; snapshot will be taken" followed by EVE-OS doing the snapshot and then calling an API to tell the app instance "snapshot was done".

Approaches we can take to implement application-consistent snapshots

At the moment we see bottom-up (Controller-initiated) and top-down (VM-initiated) models to address the problem. And you need to understand that both approaches are important and have the right to exist depending on user stories. In fact, based on user stories, and regardless of whether we will implement one or another approach or even both at once, we need to have our own or already existing guest agent that will work on the VM side (the reasons will become clearer a little later).

VM-initiated snapshot

In the **VM-initiated snapshot,** an application running in the VM would ensure it flushed the latest states on the disk, and handshake to the Local Profile Server to initiate the snapshot. This is the most straightforward approach for us, but has its drawbacks - once the flush operation is completed, the system is allowed to continue write operations. This means that once we rolled back to one of the snapshots, the filesystem would likely have to use its fault-tolerance the mechanism as if a sudden power-off happened. But at least, it will be guaranteed that it has the latest data from the application.

Let's try to take a closer look at the list of steps for executing a command to create or rollback a snapshot in this approach. It is also worth noting that a guest agent (for example, some future eve-guest-agent) must be running on the VM in order for the user to initiate the creation or rollback of a snapshot from the VM. Why eve-guest-agent? Because in order to implement this approach, there is a critical need to receive feedback from EVE. Thus, it will be either a completely new guest agent tailored for work and needs with EVE OS, or a modified one (for example, based on qemu-ga, google-ga, or others)

Steps:

1. The user must use the functionality of the application that runs on the VM and execute a command that will flush the cache of this application to the "physical" disk and completely end or suspend I/O in the application (if creating a snapshot, you can only suspend I/O if rollback to a snapshot, then you can complete all current write operations). In this case, everything depends on the functionality of the N application that runs on the VM. As a last resort, if the application does not support such functionality for creating snapshots/backups, then the user can always exit the application. The user will perform this step manually (or through a script) since we cannot adapt to any specific applications.
2. After the successful completion of step 1, the user, through the agent application, let it be *eve-guest-agent* for example, sends a command to create a snapshot, something like: *eve-guest-agent snapshot --create /dev/sdb*
3. The command is sent to EVE, and all the necessary conditions for creating/rolling back a snapshot are calculated on EVE and if:
    a. For example, if there is not enough free space, EVE will return an error on the eve-guest-agent VM about not having enough space to create a snapshot on EVE. There may also be another error that appeared in preparation for the requested operation. And the command is interrupted.
    b. Next step

4. EVE sends a command to the VM to do a sync and freeze the available filesystems.
5. EVE checks that the filesystems on the VM have been frozen.
6. EVE is running a command to create/rollback a snapshot
    a. If something went wrong, EVE sends an FS thaw command to the VM, and eve-guest-agent returns an error to the user
    b. Next step
7. EVE thaws the FS on the guest VM and checks that it has been thawed.
8. EVE notifies the user via eve-guest-agent that the operation was successful
9. Guest VM returns to a normal state (user returns the application to its normal state)

You can read more about this in the EVE Guest Agent proposal.

Controller-Initiated snapshot

For **Controller-Initiated snapshot**, we, as in the case of VM-initiated snapshot, must have an agent running as a daemon on the VM. And there is no way around it. The only difference will be that in place with the create or rollback command, we will also have to send a list/script with commands for the guest VM from the controller. This list of commands will also be set by the user, and it is assumed that these will be ordinary commands for running on the VM, which will be performed before and after snapshot operations to ensure consistent snapshots.

Qemu-guest-agent fsfreeze command

Also, qemu comes with the qemu-guest-agent software, which is available for Linux and Windows. The agent runs as a daemon and communicates with the host via the virtio or AF_VSOCK serial port. Qemu-guest-agent allows the VM side to execute sync/fsfreeze/fs-thaw or any other command sent by the host. This can be considered as a potential solution for the Controller-Initiated snapshot approach, or as a basis for the development of eve-guest-agent

First, EVE will send user commands to the VM to put the application into for create snapshot/backup state. After the command is successfully executed, EVE will begin the snapshot creation or rollback procedure. Next EVE send fsfreeze command

Once such a command is received by the daemon, it would flush the guest file system and temporarily freeze it. That would ensure that the snapshot is consistent and once we are rolled back to it, there will be no fsck problems.

From the guest side, this is achieved with the FIRFREEZE syscall. A subsequent FITHAW will allow the writes again.

The steps for this are rough as follows:

1. Receive a command from the controller to create or rollback a snapshot;
2. EVE sends a command to qemu-guest-agent to put the application into for create snapshot/backup state;
3. qemu-guest-agent executes some command or script;
4. EVE checking the state of the file system in the VM;
5. EVE checking sure the file system is working normally;
6. EVE checking flush RAM to disk and freeze the file system in the VM;
7. EVE checking sure the file system in the VM is frozen;
8. EVE create a new snapshot or rollback to an existing snapshot;
9. After completing the command for snapshot, EVE send the command to unfreeze the file system in the VM;
10. EVE checking sure that the file system is unfrozen and works normally;
11. EVE sends a command to qemu-guest-agent to put the application into a normal state.
12. qemu-guest-agent executes some command or script;
13. EVE sends up-to-date information to the controller.
14. VM goes back to normal updating of their volume.

**Between step-6 and step-10 the guest can not do updates.*

A similar process can be implemented through the EVE Guest Agent, which also will allow you to support work with snapshots and with the initiator - VM.

Implementing a plan of snapshot functionality in EVE

The implementation of snapshots in EVE is divided into two parts.

In the first part, the API and the main functionality for processing commands for snapshots are implemented, namely:

- **Create**
  To create an EVE snapshot, it is enough to receive from the controller a configuration for this snapshot, which did not exist before. The configuration for this command must include the UUID of the logical volume for which the snapshot will be created, the UUID of the snapshot itself (generated on the controller side, but it can also be generated in EVE if it is not present), DisplayName is the visual user-friendly name of the snapshot on the controller (*Can be obtained from the controller when created, or it is maybe generated automatically in EVE*). This command also requires actions with the application (*Dumping data to disk, which is written above, or completely suspending it in a simple implementation to avoid problems*)
- **Remove**
  Remove snapshot command is executed when the controller does not receive a snapshot configuration that exists in EVE.
- **Rename**
  This command has no effect on ZFS. Changes the Display-name for a snapshot in EVE. Executed when the display-name the field is changed in the configuration received from the controller.
- **Rollback**
  This command is executed when the counter is changed in the incoming configuration for a specific snapshot.
  This command also requires preparatory actions on the EVE side (*For example, a complete stop of the application or actions to interact with the application to reset the data in the application to the logical volume, this process is*

described in the previous subchapter). Another thing to consider is how snapshot rollback works in ZFS, that the rollback to snapshot k command will implicitly automatically delete all snapshots from k+1 to N if the deletion of snapshots from k+1 to N was not initiated by the controller prior to the rollback command. For this reason, before rolling back to snapshot k, it is recommended to delete snapshots from k+1 to N (*automatically*) on the controller.

To implement these commands on the EVE side, the libzfs library is used.

In the second part of the implementation, it is planned to implement a process for EVE to interact with applications to flush data and suspend I/O to disks inside the application before executing commands to Create or Rollback a Snapshot. Proposals for the implementation of this functionality have been described above. At the moment, before executing these two commands, it is necessary to turn off the application on the side of the controller in order to avoid problems with data corruption.
And the task of the second part of the implementation is to change this, to simplify the processes of working with snapshots without the need to suspend applications.

Snapshot information we expect from the controller

To manage snapshots in EVE through a controller, a new configuration message **SnapshotConfig** is planned to be added:

```
// RollbackCmd - snapshot rollback command
//
// If the counter in the DeviceOpsCmd is different, it will cause a
// rollback operation.
//
// The fields from RollbackCmd (snapshot_uuid, volume_uuid) will
// always match the fields from the SnapshotConfig (snapshot_uuid, volume_uuid)
// to which RollbackCmd belongs.
//
// For example: take the situation where the controller requests one or more
// snapshots (uuid=1 for volume_uuid=X, then add a second SnapshotCmd
// with uuid=2 for volume_uuid=X), and then later it send a RollbackCmd with
// snapshot_uuid=1
//
// In this case, the following will happen:
// It is assumed that the RollbackCmd message cannot come separately from
// the configuration, and will always be part of the SnapshotConfig for
// every snapshot. And we will always receive from the controller a complete
// list of SnapshotConfig for all snapshots that the controller knows about.
// Then after the snapshot with uuid=2 has been created, we can also get
// a SnapshotConfig for uuid=1, where the counter in RollbackCmd will be +1.
// Thus, we will roll back to the state of the logical volume at the time
// of the snapshot with uuid=1, and the snapshot with uuid=2 will be marked
// in the info message how IMPLICITLY_DELETED (If the controller did not take
// care of deleting the snapshot with uuid=2 before the rollback command).
message RollbackCmd {
          // UUID of the snapshot to rollback to. Is the snapshot_uuid
           // from the SnapshotConfig and the real and unique name of the
          // snapshot in ZFS. This is the rollback target for volume_uuid.
           string snapshot_uuid = 1;
           string volume_uuid = 2;    // UUID of the volume to rollback
        DeviceOpsCmd rollback = 3; // Counter for rollback
}

// SnapshotConfig describes a snapshot for a specific logical
// volume that must exist on the device. It has a required field
// volume_uuid for which it was created and UUID for snapshot name in ZFS.
message SnapshotConfig {
   // The real name of the snapshot in ZFS. It is the link between
   // the command and the response to the command. It is assumed
   // that the field will always be filled with a unique value
   // that the controller will generate.
   string snapshot_uuid = 1;
   // Display name (User-friendly name). This name does not affect
   // the snapshot properties in ZFS in any way.
   // Can be filled on the controller side.
   // If a snapshot has already been created and this field has changed,
   // it can be assumed that the friendly name for this snapshot has
   // been renamed on the controller side.
   string display_name = 2;
   // Volume ID of the volume this snapshot belongs to. The field is
   // required for all messages. Must always be filled in on the
   // controller side before sending to create snapshot command.
   string volume_uuid = 3;
   // An "optional" snapshot rollback operation command, because
   // not all SnapshotConfig require a rollback command
   // If the RollbackCmd message is not enabled, it means that
   // a rollback is not requested. If RollbackCmd is enabled in this
   // config, then a rollback has been requested, and also the counter
   // in RollbackCmd will be checked.
   // It should also be taken that the rollback cmd
   // to snapshot k will implicitly automatically delete
   // all snapshots from k+1 to N, If the deletion of snapshots
   // from k+1 to N was not initiated by the controller before
   // the rollback cmd.
   RollbackCmd rollback = 4;
}
```

Information about snapshots to send to the controller

In addition to commands, it is also planned to send information about snapshots to the controller, which will be divided into two parts: Information and metrics. The need to send metrics comes from the fact that over time when data on the disk changes or when earlier snapshots are deleted, snapshots tend to increase in size, so it is necessary to inform the controller about up-to-date information about the space occupied by snapshots in ZFS.

The structure of the informational message ZInfoSnapshot will consist of the following fields:

```
// Snapshot states
enum ZSnapshotState {
          Z_SNAPSHOT_STATE_UNSPECIFIED = 0;
          // This state is used when a snapshot is in the process of being
          // created or an error occurred during the first attempt to create it.
          // (For example, the operation was delayed)
          Z_SNAPSHOT_STATE_CREATING = 1;
          // This state is used when the snapshot has been successfully created.
          Z_SNAPSHOT_STATE_CREATED = 2;
          // This state is used when the snapshot is pending deletion or
          // the first deletion attempt was not successful. Once a snapshot is deleted,
          // EVE will no longer create a ZInfoSnapshot for it.
          Z_SNAPSHOT_STATE_DELETING = 3;
          // This state is used to send information to the controller about a
          // snapshot that was implicitly deleted after a rollback snapshot
          // or volume delete command. After sending a message with this status once,
          // EVE no longer expects to receive the configuration (SnapshotConfig)
          // for this snapshot. If the controller continues to send configuration
          // for a snapshot that was implicitly deleted, then EVE will ignore
          // the configuration for that snapshot and resend the ZInfoSnapshot
          // message to the controller with the status IMPLICITLY_DELETED.
          Z_SNAPSHOT_STATE_IMPLICITLY_DELETED = 4;
}

// ZInfoSnapshot - Information about snapshot in zfs for zvol
message ZInfoSnapshot {
          google.protobuf.Timestamp creation_time = 1;
          string snapshot_uuid = 2;           // Link a command and a response. Is the real name of the
snapshot in ZFS
          string volume_uuid = 3;             // Volume ID of the volume this snapshot belongs to.
          string display_name = 4;            // Ex: "Tuesday" or creation time (User-friendly name)
          bool encryption = 5;                // This is just an indication. Inherited from zvol.
          ZSnapshotState current_state = 6;   // Displays the current state of the snapshot
          ErrorInfo error_msg = 7;            // Ops error
          uint32 rollback_cmd_counter = 8;    // Counter for rollback cmd from the DevOpsCmd in RollbackCmd
          google.protobuf.Timestamp rollback_time_last_op = 9;  // The time when the last rollback operation
was performed for this snapshot
}
```

The structure of the message ZMetricSnapshot with metrics will consist of the following fields:

```
// Metrics for a snapshot
// When a snapshot is created, its disk space is initially shared between
// the snapshot and the file system, and possibly with previous snapshots.
// As the file system changes, disk space that was previously shared becomes
// unique to the snapshot, and thus is counted in the snapshot's used property.
// Additionally, deleting snapshots can increase the amount of disk space
// unique to (and thus used by) other snapshots, however, this does not increase
// the amount of disk space specified for the volume itself.
message ZMetricSnapshot {
          // Snapshot UUID
           string snapshot_uuid = 1;
           // User-friendly name on controller
           string display_name = 2;
           // The used space of a snapshot is space that is referenced exclusively
           // by this snapshot. If this snapshot is destroyed, the amount of used
           // space will be freed. Space that is shared by multiple snapshots isn't
           // accounted for in this metric. When a snapshot is destroyed, space that
           // was previously shared with this snapshot can become unique to snapshots
           // adjacent to it, thus changing the used space of those snapshots.
           // The used space of the latest snapshot can also be affected by changes
           // in the file system. Note that the used space of a snapshot is a subset
           // of the written space of the snapshot. (in bytes)
           uint64 used_space = 3;
           // Identifies the amount of data accessible by this snapshot, which might
           // or might not be shared with other datasets in the pool. When a snapshot or
           // clone is created, It initially references the same amount of space as the
           // file system or snapshot it was created from because
           // its contents are identical. (in bytes)
           uint64 referenced = 4;
           // Identifies the compression ratio achieved for this snapshot,
           // expressed as a multiplier.
           double compressratio = 5;
           // Specifies the logical size of the volume this snapshot belongs to (in bytes)
           uint64 vol_size = 6;
           // The amount of space that is "logically" accessible by this dataset.
           // See the referenced property. The logical space ignores the effect of
           // the compression and copies properties, giving a quantity closer to
           // the amount of data that applications see.
           // However, it does include space consumed by metadata. (in bytes)
           uint64 logicalreferenced = 7;
           // The amount of referenced space written to this dataset since the specified
           // snapshot. This is the space that is referenced by this dataset but was
           // not referenced by the specified snapshot. (in bytes)
           uint64 written = 8;
}
```

Comments on API messages from these blocks can be left directly in PR #2633 at this link.

Comments on the implementation for processing these commands can also be left in PR #2607 at this link

Discussion

It is necessary to define a clear course of action for implementing the second part, namely adding an implementation for EVE to interact with applications to perform certain actions within the application, before executing commands to create or roll back snapshots in EVE. Implementations have been suggested in this document, but this is not the final path and is still under discussion.

References

Storage in EVE

Feature Roadmap

PR: Add an API to support snapshots in ZFS #2633

PR: Add implementation of functionality for working with snapshots #2607

EVE Guest Agent

QEMU Guest Agent