Managing Batch Processing

Alan Smith

SENIOR CONSULTANT – ACTIVE SOLUTION SWEDEN

@alansmith www.cloudcasts.net

Overview

Handling Failed Tasks

Demo: Handling Failed Tasks

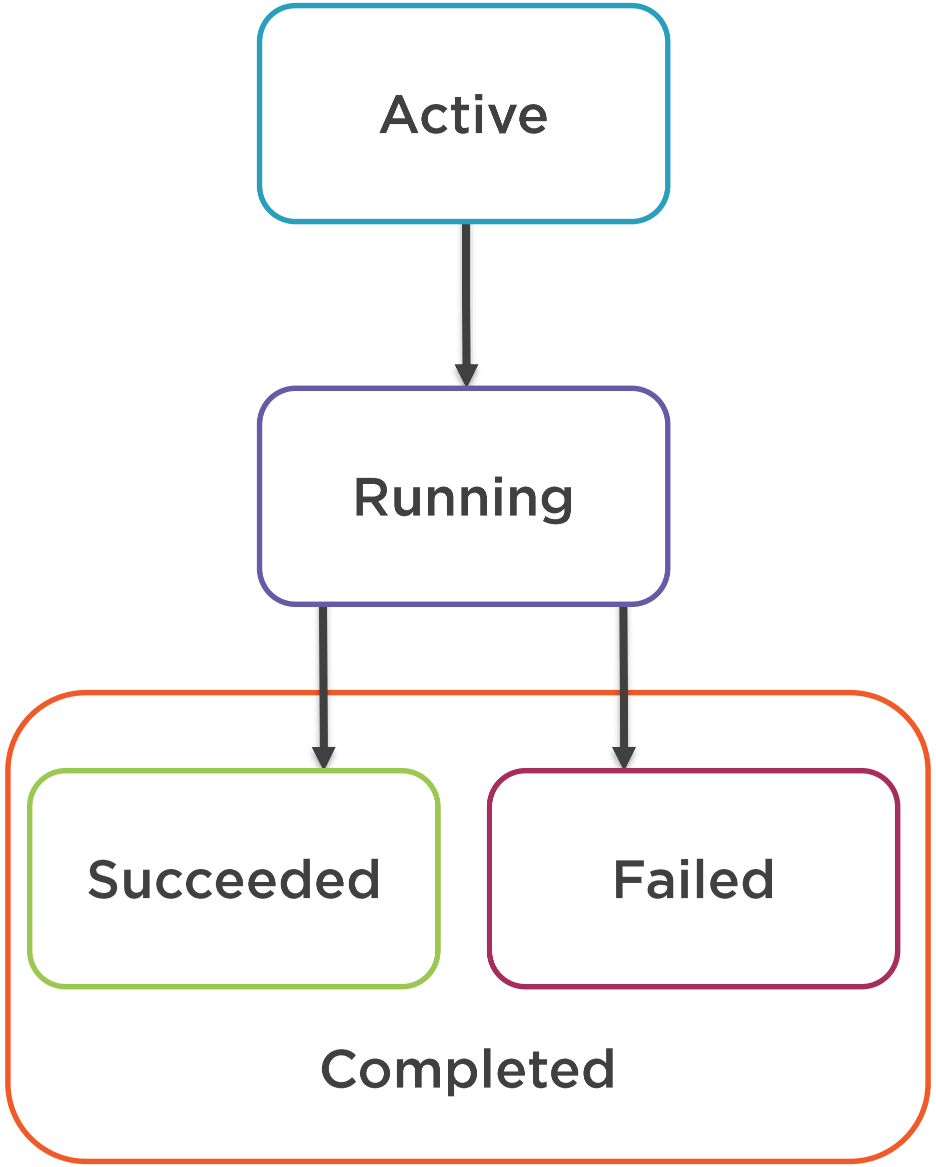
Optimizing Job Processing

Demo: Optimizing Job Processing

Managing Scalability and Auto-Scaling

Demo: Configuring Auto-Scaling

Handling Failed Tasks



Task Lifecycle

Active

- Tasks that are ready to run
- Dependency statuses are met

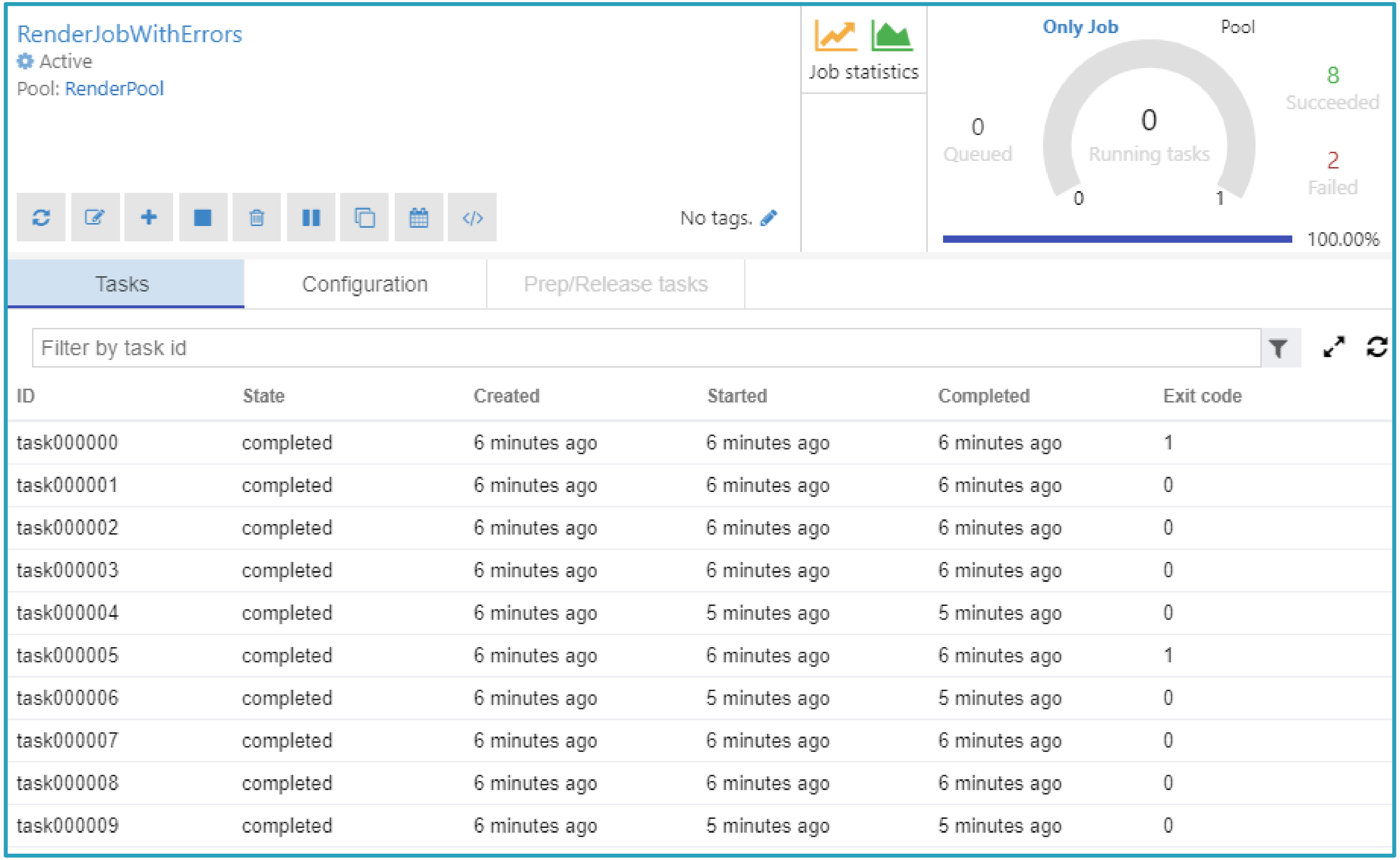
Running

- Node is actively processing task

Completed

- Succeeded
    - Task has successfully completed
- Failed
    - Task did not complete successfully

Viewing Error Statistics

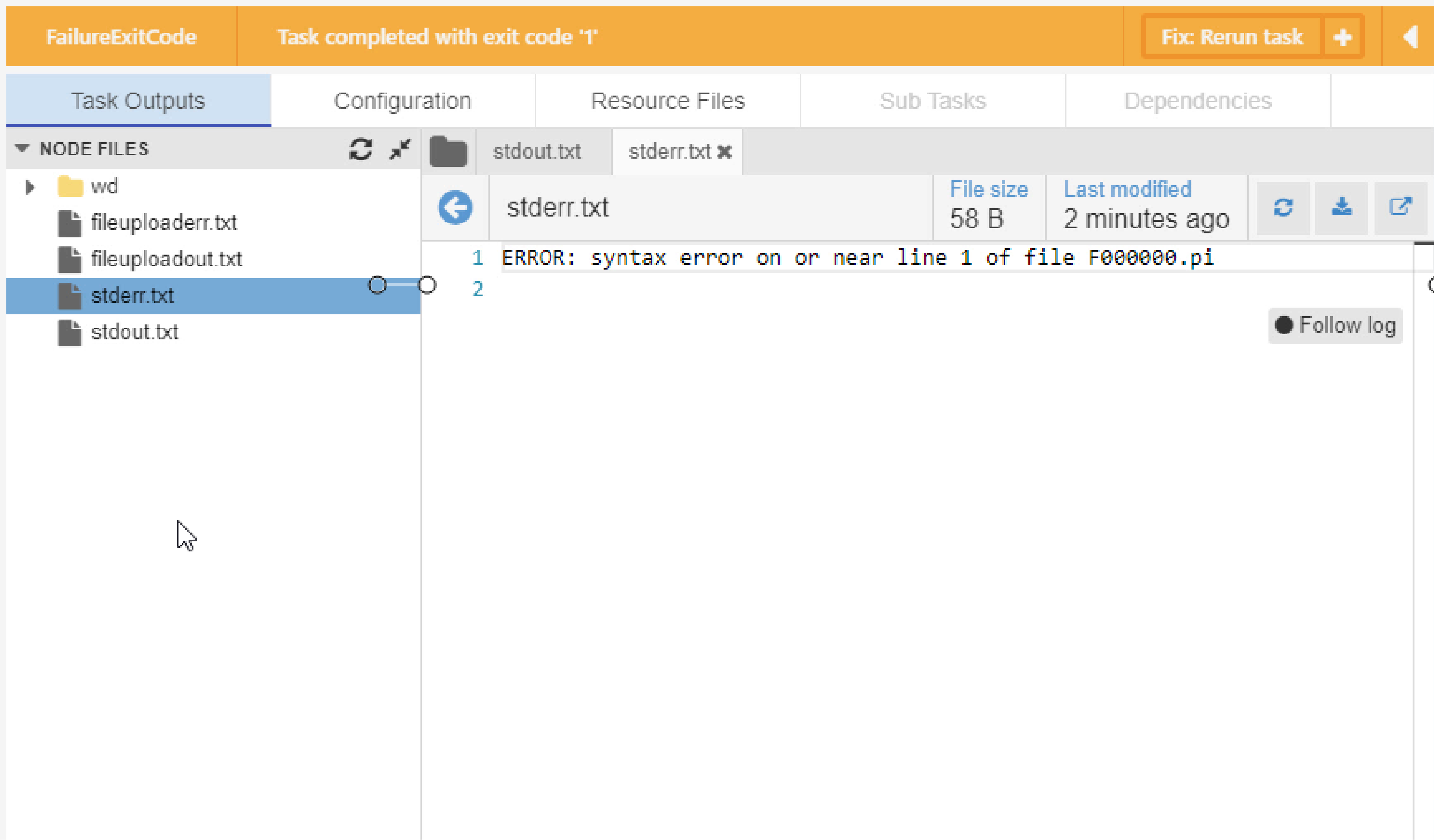


RenderJobWithErrors
Active
Pool: RenderPool

No tags.

ID	State	Created	Started	Completed	Exit code
task000000	completed	6 minutes ago	6 minutes ago	6 minutes ago	1
task000001	completed	6 minutes ago	6 minutes ago	6 minutes ago	0
task000002	completed	6 minutes ago	6 minutes ago	6 minutes ago	0
task000003	completed	6 minutes ago	6 minutes ago	6 minutes ago	0
task000004	completed	6 minutes ago	5 minutes ago	5 minutes ago	0
task000005	completed	6 minutes ago	6 minutes ago	6 minutes ago	1
task000006	completed	6 minutes ago	5 minutes ago	5 minutes ago	0
task000007	completed	6 minutes ago	6 minutes ago	6 minutes ago	0
task000008	completed	6 minutes ago	6 minutes ago	6 minutes ago	0
task000009	completed	6 minutes ago	5 minutes ago	5 minutes ago	0

Viewing Error Details



```
ERROR: syntax error on or near line 1 of file F000000.pi
```

```
TaskCounts taskCounts = await m_BatchClient.JobOperations.GetJobTaskCountsAsync(jobId);

int numberOfFailedTasks = taskCounts.Failed;
```

Retrieving failed task count using code

```
var tasks = m_BatchClient.JobOperations.ListTasks(jobId);

foreach (var task in tasks)
{
    if (task.ExecutionInformation.ExitCode != 0)
    {
        Console.WriteLine($"Task { task.Id } failed.");
    }
}
```

Retrieving failed tasks

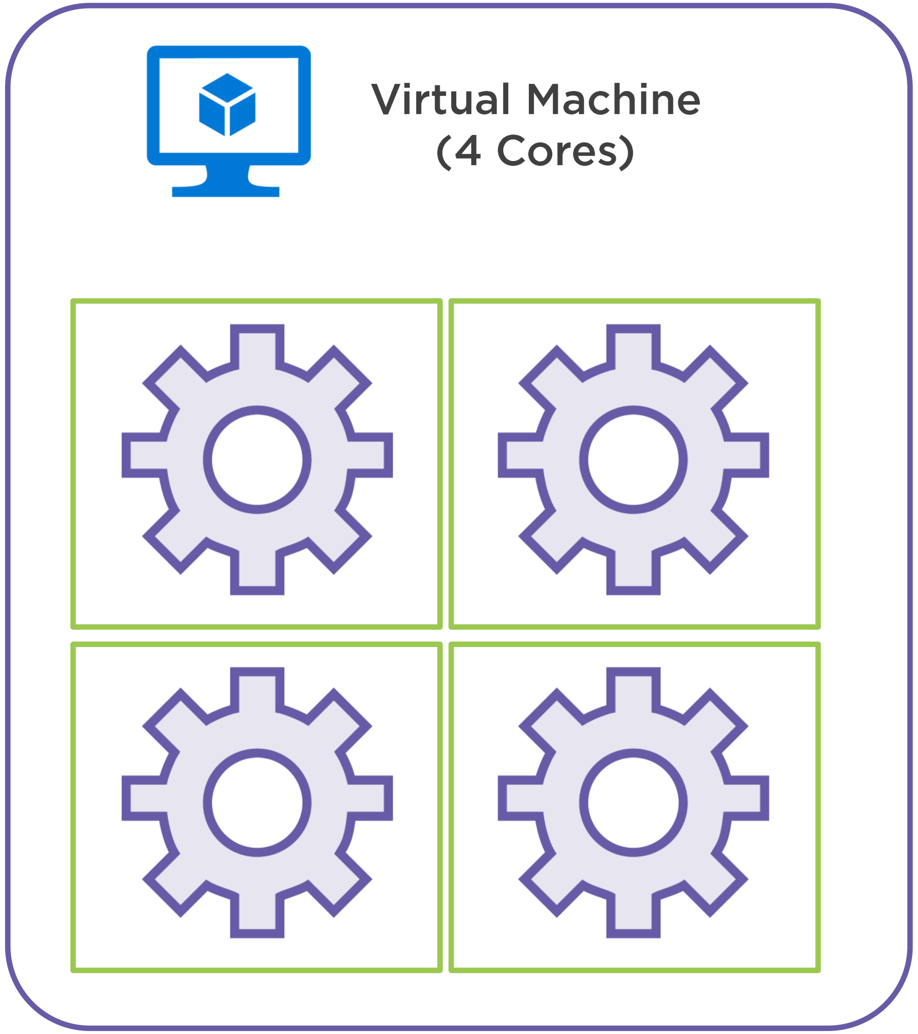
Demo

Handling Failed Tasks

- Generating tasks that will fail
- Analyzing task execution
- Detecting failed tasks
- Diagnosing and fixing errors
- Rerunning failed tasks

Optimizing Job Processing

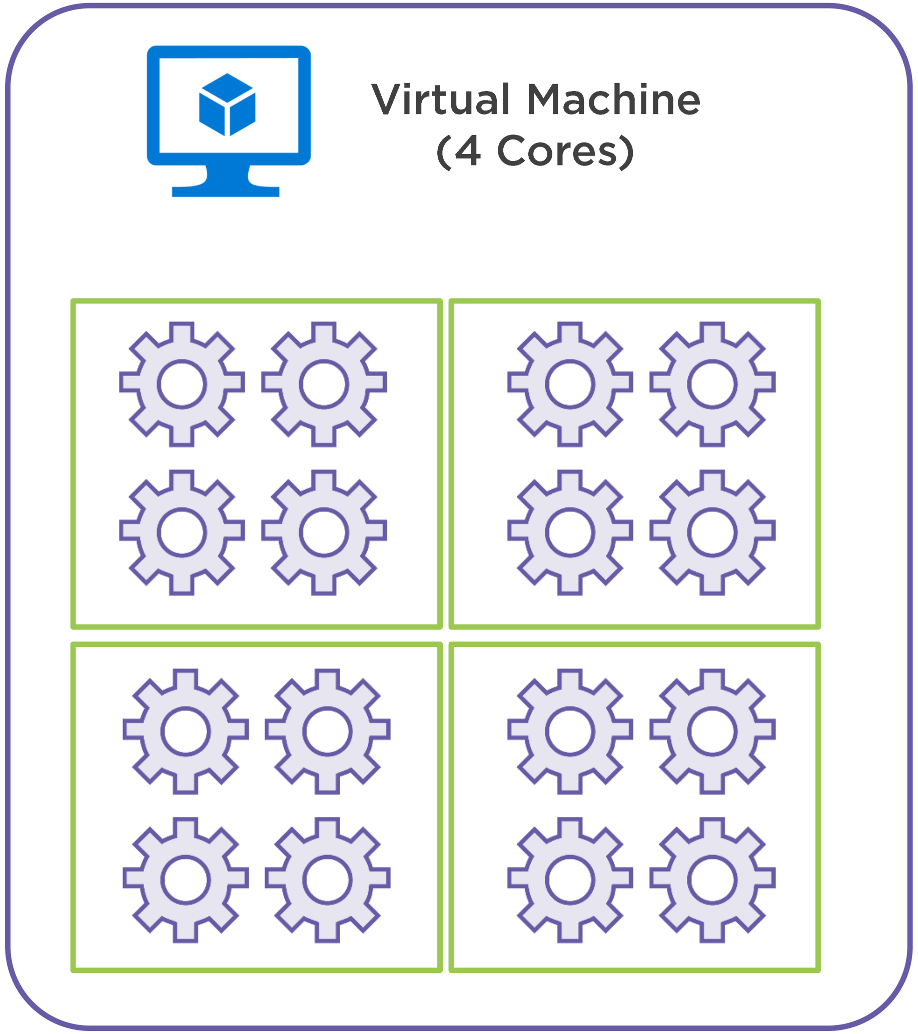
Max Tasks Per Node



Run multiple tasks on nodes

- Optimize resource usage

Max Tasks Per Node



Run multiple tasks on nodes

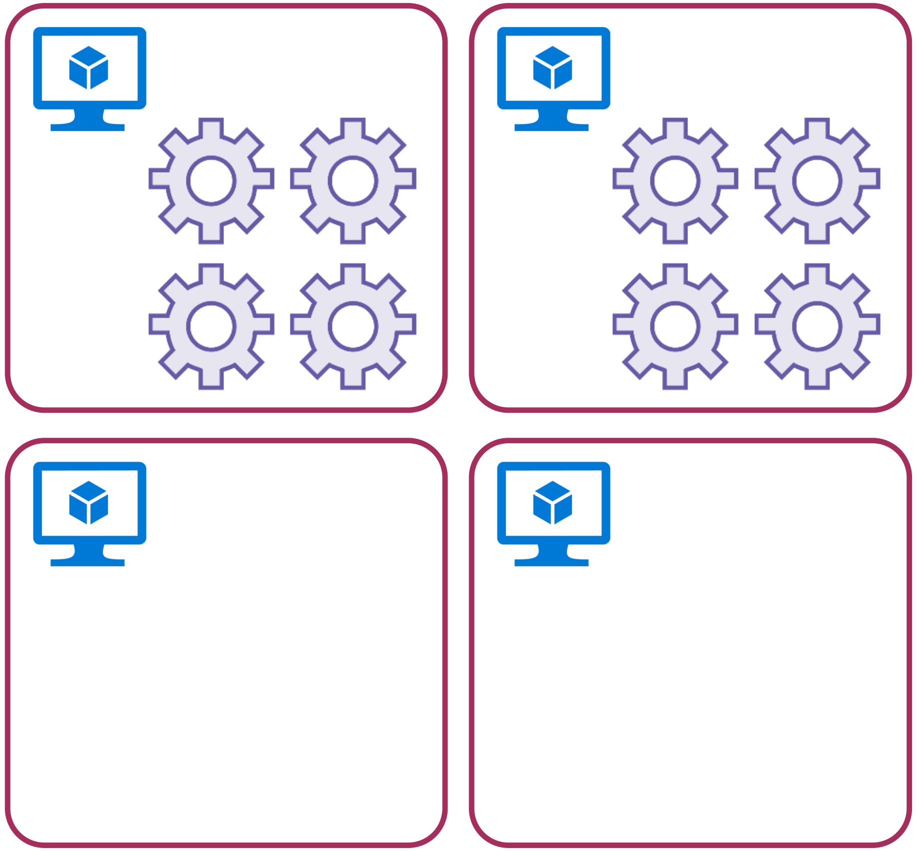
- Optimize resource usage

Up to 4 x Core Count of Node

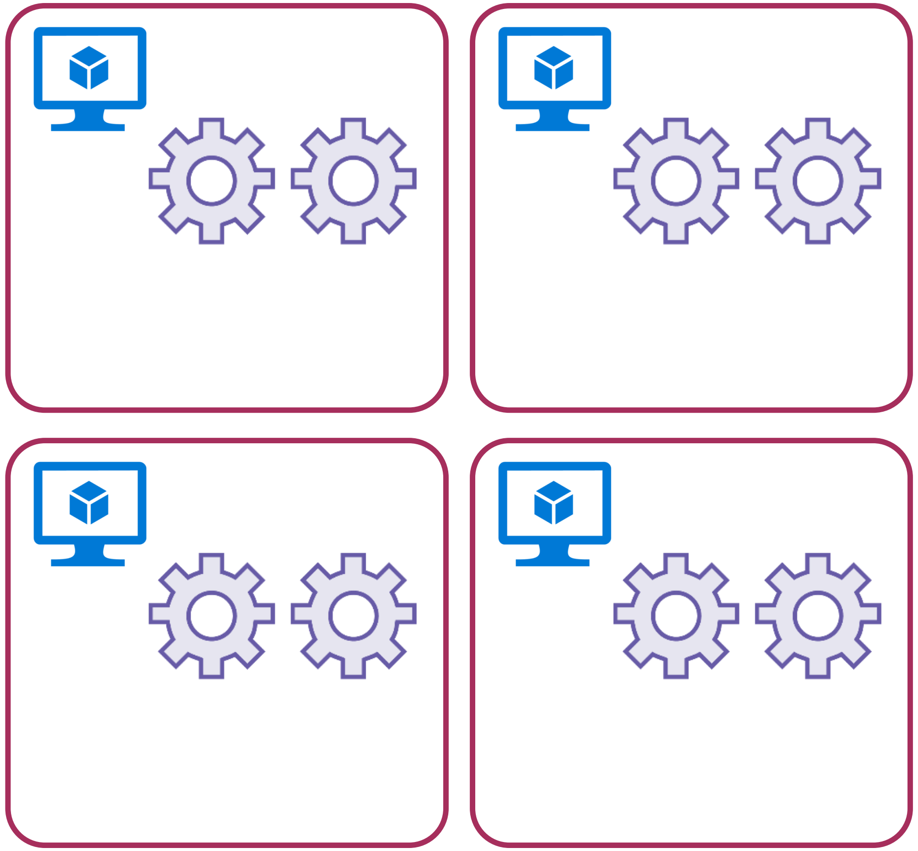
Task Scheduling Policy

- Spreading, distribute across nodes
- Packing, minimize node utilization

Task Scheduling Policy



Packed



Spread

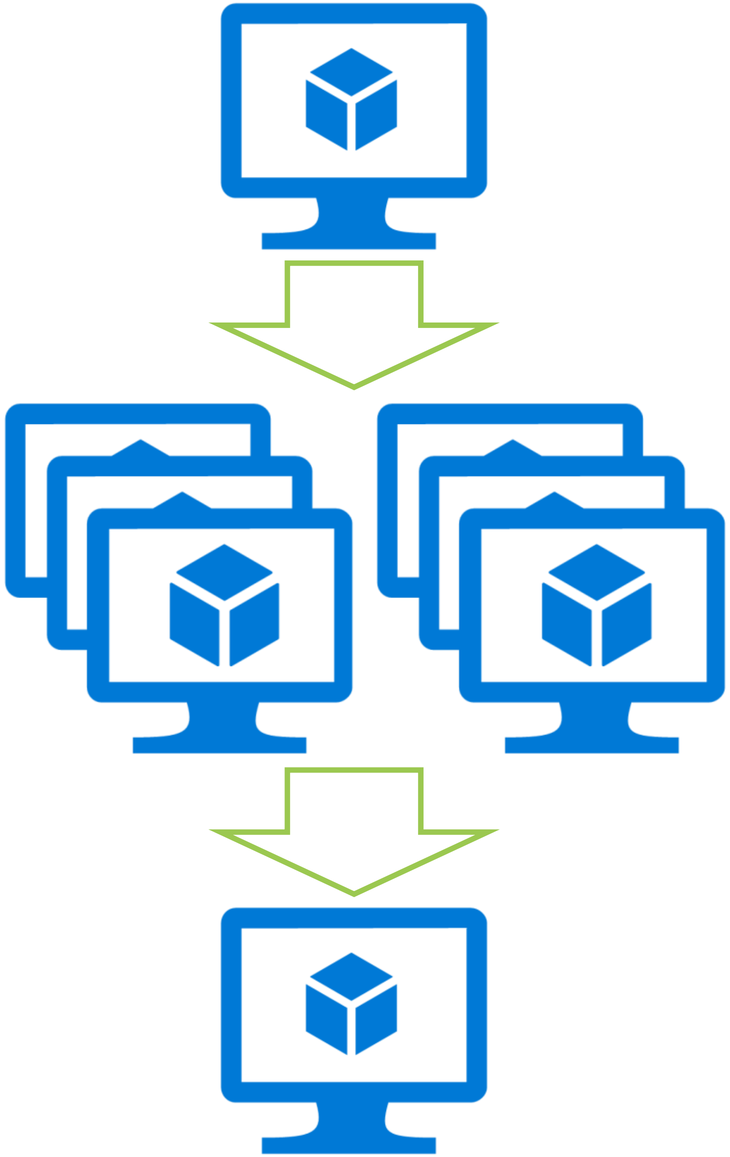
Demo

Optimizing Job Processing

- Provisioning multi-core VMs
- Running multiple concurred tasks on a node
- Setting task scheduling policy

Managing Scalability and Auto-Scaling

Auto-Scaling

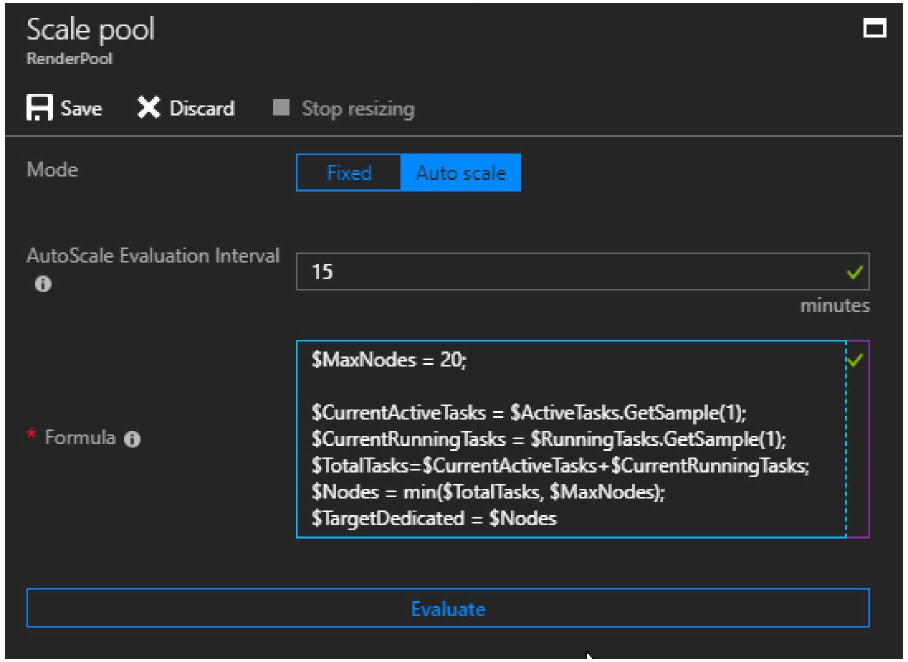


Set the number of nodes in a pool

Based on service variables

- CPU percentage
- Memory
- Number of tasks
  - Active
  - Running
  - Pending
  - Failed

Auto-Scale Formula



Mode: Fixed [✓ Auto scale]

AutoScale Evaluation Interval: 15 minutes

* Formula

```
$MaxNodes = 20;

$CurrentActiveTasks = $ActiveTasks.GetSample(1);
$CurrentRunningTasks = $RunningTasks.GetSample(1);
$TotalTasks=$CurrentActiveTasks+$CurrentRunningTasks;
$Nodes = min($TotalTasks, $MaxNodes);
$TargetDedicated = $Nodes
```

Retrieve task and resource metric data

Adjust pool size based on that data

Comprised of up to 100 statements

Uses service-defined and user-defined variables

Common Service Defined Variables

Read-Only

- $CPUPercent
- $MemoryBytes
- $ActiveTasks
- $RunningTasks
- $PendingTasks
- $SucceededTasks
- $FailedTasks

Read-Write

- $TargetDedicatedNodes
- $TargetLowPriorityNodes
- $NodeDeallocationOption
  - requeue
  - terminate
  - taskcompletion
  - retaineddata

```
doubleVec GetSample(count);
doubleVec GetSample(startTime);
doubleVec GetSample(startTime, endTime);

// Get last available sample for active (queued) tasks.
runningTasksSample = $ActiveTasks.GetSample(1);

// Get pending task samples for the past 15 minutes.
runningTasksSample =
    $ActiveTasks.GetSample(1 * TimeInterval_Minute, 15 * TimeInterval_Minute);
```

Sampling Ranges of Metrics

Resource metrics sampled every 30 seconds

Metrics sampled by formula may be delayed or missing

Sample metrics over a time period, starting from 1 minute

```
$MaxNodes = 20;

$CurrentActiveTasks = $ActiveTasks.GetSample(1 * TimeInterval_Minute, 6 * TimeInterval_Minute);

$CurrentRunningTasks = $RunningTasks.GetSample(1 * TimeInterval_Minute, 6 * TimeInterval_Minute);

$TotalTasks=$CurrentActiveTasks+$CurrentRunningTasks;

$Nodes = min($TotalTasks, $MaxNodes)

$TargetDedicated = $Nodes
```

Auto-scaling based on queue length

Demo

Configuring Auto-Scaling

- Defining an auto-scaling formula
- Configuring auto-scaling
- Testing and analyzing auto-scaling

Summary

Completed tasks can be succeeded or failed

Failed tasks have non-zero exit code

Output files can be used to determine cause of task failures

Multiple concurrent tasks can be run on each node in a pool

Auto-scaling can be applied to a pool

Auto-scaling formula used to set node count based on resource and task details