# StorageVirtualNode Class Documentation

Cloud SaaS Simulation Component

Engr.Daniel Moune

ICT University, Yaounde, Cameroon

July 30, 2025

# StorageVirtualNode Class

## Core Responsibilities

- Simulates cloud storage node with:
  - Storage capacity tracking
  - Network bandwidth management
  - File transfer operations
- Implements chunk-based file transfers
- Provides resource utilization metrics

## Key Attributes

- `node_id`
- `total_storage`
- `used_storage`
- `bandwidth`

## Data Structures

- `active_transfers`
- `stored_files`
- `connections`

## __init__ Method - Code

```python
def __init__(self, node_id: str,
             cpu_capacity: int,
             memory_capacity: int,
             storage_capacity: int,
             bandwidth: int):
    # Resource capacities
    self.node_id = node_id
    self.total_storage = storage_capacity * 1024**3
    self.bandwidth = bandwidth * 10**6

    # Utilization tracking
    self.used_storage = 0
    self.network_utilization = 0

    # Data structures
    self.active_transfers = {}
    self.stored_files = {}
    self.connections = {}
```

## __init__ Method - Explanation

### Key Features

- Converts human-friendly units to bytes/bits:
    - GB $\rightarrow$ bytes ($1024^3$)
    - Mbps $\rightarrow$ bits/sec ($10^6$)
- Initializes empty state tracking:
    - Storage utilization
    - Network utilization
    - Active transfers

### Design Considerations

- All resources tracked in base units (bytes/bits)
- Dictionaries used for O(1) lookups
- Separate tracking of allocated vs used resources

## initiate_file_transfer - Code

```python
def initiate_file_transfer(self, file_id: str,
                           file_name: str,
                           file_size: int,
                           source_node: str = None):
    # Check storage availability
    if self.used_storage + file_size > self.total_storage:
        return None

    # Create transfer record
    chunks = self._generate_chunks(file_id, file_size)
    transfer = FileTransfer(
        file_id=file_id,
        file_name=file_name,
        total_size=file_size,
        chunks=chunks
    )
    self.active_transfers[file_id] = transfer
    return transfer
```

## initiate_file_transfer - Explanation

### Workflow

1. Validates available storage space
2. Splits file into chunks (calls `_generate_chunks`)
3. Creates transfer record
4. Adds to active transfers dictionary

### Key Parameters

| | |
|---|---|
| `file_id` | Unique transfer identifier |
| `file_size` | In bytes |
| `source_node` | Optional origin node |

### Edge Cases

- Returns None if insufficient space
- Handles duplicate file_id (overwrites)

## Chunk Generation - Code

```python
def _calculate_chunk_size(self, file_size: int) -> int:
    # Adaptive chunk sizing
    if file_size < 10 * 1024**2:     # < 10MB
        return 512 * 1024            # 512KB
    elif file_size < 100 * 1024**2: # < 100MB
        return 2 * 1024**2           # 2MB
    else:
        return 10 * 1024**2          # 10MB

def _generate_chunks(self, file_id: str,
                     file_size: int) -> List[FileChunk]:
    chunk_size = self._calculate_chunk_size(file_size)
    return [
        FileChunk(
            chunk_id=i,
            size=min(chunk_size, file_size - i*chunk_size),
            checksum=hashlib.md5(f"{file_id}-{i}".encode()
        ).hexdigest()
        for i in range(math.ceil(file_size/chunk_size))
    ]
```

# Chunk Generation - Explanation

## Adaptive Chunk Sizing

- Smaller chunks for small files
- Larger chunks for better throughput on big files
- Thresholds configurable

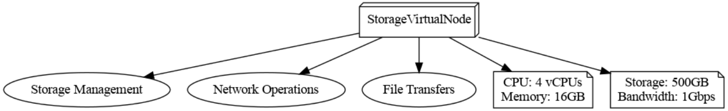
## Chunk Features

- Each chunk gets:
  - Sequential ID
  - Actual size (handles partial chunks)
  - MD5 checksum
- List comprehension for efficient generation

## Optimization

- Chunk size affects:
  - Network efficiency
  - Memory usage

## Architecture Summary



### Key Strengths

- Realistic resource modeling
- Network-aware transfers
- Detailed monitoring
- Extensible design