

What is a Module?

Introduction

Modules are the foundation of PUQcloud's extensible billing system architecture. They are self-contained, professionally developed components that seamlessly extend the platform's functionality. Each module can add new service types, integrate with external APIs, implement payment gateways, or provide communication channels - all while maintaining consistent performance, security, and user experience standards.

Think of modules as professional plugins that transform PUQcloud from a basic billing system into a comprehensive service management platform capable of handling any business model.

Core Architecture Concept

A **module** in PUQcloud is a PHP class that inherits from one of four specialized base classes and implements specific business logic. The modular framework provides:

- **Standardized lifecycle management** (Installation, activation, updates, deactivation)
- **Built-in security and permission systems** with granular access control
- **Asynchronous task processing** for scalable operations
- **Comprehensive logging and monitoring** capabilities
- **Multi-language support** with Laravel's localization
- **Template and view management** with Blade templating
- **Database integration** with migrations and models
- **External API integration** patterns and best practices

Module System Architecture

Hierarchical Inheritance Structure

PUQcloud's module system follows a clean inheritance hierarchy:

```
App\Modules\Module (Base Class)
├─ Product      → Full service lifecycle management
├─ Plugin       → System functionality extensions
├─ Payment     → Payment gateway integrations
```

Class Responsibilities and APIs

A module is an object-oriented PHP class that inherits from `App\Modules\Module` and shares a common contract:

- **Lifecycle:** `activate()`, `update()`, `deactivate()`
- **Rendering:** `view(string $template, array $data = [])`
- **Configuration:** `config(string $key): mixed`
- **Asynchronous jobs:** queue tasks with `Task::add('ModuleJob', 'Module', $data, $tags)`
- **Logging:** `logInfo()`, `logError()`, `logDebug()`
- **Security:** built-in permission checks in admin routes/controllers

Derived Module Types and Typical Methods

Product (Service Management)

- **Product config:** `getProductData()`, `saveProductData()`, `getProductPage()`
- **Service config:** `getServiceData()`, `saveServiceData()`, `getServicePage()`
- **Lifecycle:** `create()` → `createJob()`, `suspend()` → `suspendJob()`, `unsuspend()` → `unsuspendJob()`, `termination()` → `terminationJob()`, `change_package()` → `change_packageJob()`
- **Client area:** `getClientAreaMenuConfig()`, `variables_{tab}()`, `controllerClient_{tab}{Method}()`
- **Admin area:** `adminPermissions()`, `adminSidebar()`, `adminWebRoutes()`, `adminApiRoutes()`
- **Status:** update provisioning with `Service::setProvisionStatus()`

Plugin (System Extensions)

- **Lifecycle:** initialize resources in `activate()`, clean up in `deactivate()`, evolve in `update()`
- **Admin area:** expose UI via `adminSidebar()`, `adminWebRoutes()`, `adminApiRoutes()`
- **Background work:** schedule/execute periodic tasks via the queue (`Task::add()`) and event hooks (see `hooks.php`)
- **Note:** typically no client-area UI
- **Data:** define plugin-specific tables/models as needed

Payment (Payment Processing)

- **Gateway config:** `getModuleData()`, per-gateway settings page via `getSettingsPage()`
- **Checkout UI:** render payment form/session with `getClientAreaHtml()`
- **Redirects:** provide `getReturnUrl()` and `getCancelUrl()`
- **Webhooks:** handle gateway callbacks (e.g., `apiWebhookPost()`/`apiWebhookGet()`)
- **Transactions:** log, reconcile, and handle refunds/chargebacks

Notification (Communication Channels)

- **Channel config:** `getModuleData()` (server, credentials, encryption, etc.)
- **Delivery:** `send(array $data)` with validation, retries, and logging
- **Templating:** render content using Blade templates and per-channel variables
- **Reliability:** implement rate limiting and retry/backoff policies

Standard Module Structure

Every module follows this professional directory structure:

```
modules/{Type}/{ModuleName}/
├─ {ModuleName}.php           # Main module class (required)
├─ config.php                 # Module metadata and settings (required)
├─ hooks.php                  # Event hooks and listeners (optional)
├─ Controllers/               # HTTP request handlers
│   └─ {ModuleName}Controller.php
├─ Services/                  # External API clients and business logic
│   └─ {ExternalService}Client.php
├─ Models/                    # Database models and data validation
│   └─ {ModuleName}.php
│   └─ {ModuleName}Server.php
├─ views/                     # User interface templates
│   └─ admin_area/           # Admin panel interface
│       └─ product.blade.php
│       └─ service.blade.php
│       └─ configuration.blade.php
│   └─ client_area/          # Client panel interface
│       └─ general.blade.php
│       └─ statistics.blade.php
│       └─ management.blade.php
│   └─ assets/               # Static resources
│       └─ css/
│       └─ js/
│       └─ img/
└─ lang/                      # Internationalization files
    └─ en.php
    └─ pl.php
    └─ {language}.php
```

Component Breakdown

Component	Purpose	Required	Description
Main Class	Core business logic and lifecycle management	☐ Required	Implements module functionality
Configuration	Module metadata, version, and settings	☐ Required	Defines module properties
Controllers	Handle HTTP requests and API endpoints	⚠ Conditional	Required for admin/client interfaces
Models	Database interactions and data validation	⚠ Conditional	Required for data persistence
Views	User interface templates and forms	⚠ Conditional	Required for user interaction
Services	External API clients and business logic	☐ Optional	For third-party integrations
Language Files	Multi-language support and localization	☐ Optional	For internationalization

Module Types Deep Dive

1. Product Modules ☐☐ (Service Management)

Purpose: Complete service lifecycle management with customer billing integration

Real-World Examples:

- **puqNextcloud:** Cloud storage and collaboration platform
- **VPS Services:** Virtual private server management

Key Capabilities:

- □ Full service lifecycle (create, suspend, unsuspend, terminate, upgrade)
- □ Client area integration with custom tabs and interfaces
- □ Automated billing and invoicing integration
- □ Resource provisioning through external APIs
- □ Asynchronous task processing for scalable operations
- □ Service status monitoring and health checks

Architecture Example (based on real puqNextcloud implementation):

```
<?php

use App\Models\Service;
use App\Models\Task;
use App\Modules\Product;

class NextcloudService extends Product
{
    // Product configuration for admin
    public function getProductData(array $data = []): array
    {
        $this->product_data = [
            'group_uuid' => $data['group_uuid'] ?? '',
            'username_prefix' => $data['username_prefix'] ?? 'nc_',
            'username_suffix' => $data['username_suffix'] ?? '',
            'quota' => $data['quota'] ?? '1GB',
        ];
        return $this->product_data;
    }

    // Service instance creation
    public function create(): array
    {
        $data = [
            'module' => $this,
            'method' => 'createJob',
            'tries' => 1,
            'backoff' => 60,
            'timeout' => 600,
            'maxExceptions' => 1,
        ];
    }
}
```

```

    $service = Service::find($this->service_uuid);
    $service->setProvisionStatus('processing');
    Task::add('ModuleJob', 'Module', $data, ['create']);

    return ['status' => 'success'];
}

// Asynchronous provisioning logic
public function createJob(): array
{
    try {
        $service = Service::find($this->service_uuid);

        // Generate unique credentials
        $this->service_data['username'] = $this->product_data['username_prefix']
            .
                random_int(100000, 999999)
            .
                $this->
>product_data['username_suffix'];
        $this->service_data['password'] =
generateStrongPassword(10);

        // Provision through external API
        $apiClient = new NextcloudAPIClient($this->getServerConfig());
        $result = $apiClient->createUser($this->service_data);

        if ($result['status'] === 'success') {
            $service->setProvisionStatus('completed');
            return ['status' => 'success'];
        } else {
            $service->setProvisionStatus('failed');
            return ['status' => 'error', 'message' => $result['error']];
        }
    } catch (Exception $e) {
        $service->setProvisionStatus('failed');
        $this->logError('createJob', 'Provisioning failed', $e->getMessage());
        return ['status' => 'error'];
    }
}

```

```

}

// Client area interface configuration
public function getClientAreaMenuConfig(): array
{
    return [
        'general' => [
            'name' => 'Overview',
            'template' => 'client_area.general',
        ],
        'files' => [
            'name' => 'File Manager',
            'template' => 'client_area.files',
        ],
        'statistics' => [
            'name' => 'Usage Statistics',
            'template' => 'client_area.statistics',
        ],
    ];
}
}

```

2. Plugin Modules □□ (System Extensions)

Purpose: Extend system functionality without managing individual services

Real-World Examples:

- **Server Monitoring:** Infrastructure health monitoring and alerting
- **Advanced Analytics:** Custom reporting and business intelligence
- **Integration Plugins:** Synchronization with CRM, accounting systems
- **Workflow Automation:** Custom business process automation
- **Security Scanners:** Vulnerability assessment and monitoring

Key Capabilities:

- □ System-wide functionality enhancement
- □ Admin interface integration with custom pages
- □ Event-driven operations and hooks
- □ Background task scheduling and execution
- □ Custom data collection and processing
- □ Integration with external services and APIs

Architecture Example:

```
<?php

use App\Modules\Plugin;
use Illuminate\Support\Facades\Schema;

class ServerMonitoringPlugin extends Plugin
{
    public function activate(): string
    {
        try {
            // Create monitoring infrastructure
            Schema::create('server_monitoring_checks', function (Blueprint $table)
            {
                $table->id();
                $table->string('name');
                $table->string('type'); // ping, http, port, ssl
                $table->string('target'); // IP/URL to monitor
                $table->json('config'); // Check-specific configuration
                $table->integer('interval')->default(300); // Check interval in
seconds
                $table->boolean('active')->default(true);
                $table->timestamps();
            });

            Schema::create('server_monitoring_results', function (Blueprint $table)
            {
                $table->id();
                $table->foreignId('check_id')-
>constrained('server_monitoring_checks');
                $table->boolean('status'); // up/down
                $table->integer('response_time')->nullable(); // milliseconds
                $table->string('error_message')->nullable();
                $table->timestamp('checked_at');
                $table->timestamps();
            });

            $this->logInfo('activate', 'Monitoring plugin activated
successfully');
```

```

        return 'success';
    } catch (Exception $e) {
        $this->logError('activate', 'Activation failed: ' . $e->getMessage());
        return 'Error: ' . $e->getMessage();
    }
}

public function adminWebRoutes(): array
{
    return [
        [
            'method' => 'get',
            'uri' => 'dashboard',
            'permission' => 'monitoring-dashboard',
            'name' => 'dashboard',
            'controller' => 'ServerMonitoringController@dashboard',
        ],
        [
            'method' => 'get',
            'uri' => 'checks',
            'permission' => 'monitoring-checks',
            'name' => 'checks',
            'controller' => 'ServerMonitoringController@checks',
        ],
    ];
}
}

```

3. Payment Modules (Payment Processing)

Purpose: Secure payment processing through various payment gateways

Real-World Examples:

- **puqStripe:** Credit card processing through Stripe
- **puqPayPal:** PayPal payment integration
- **puqPrzelewy24:** Polish payment system integration
- **puqBankTransfer:** Bank transfer instructions and tracking
- **Cryptocurrency Payments:** Bitcoin, Ethereum, etc.

Key Capabilities:

- Secure payment form generation and processing
- Multiple currency support and conversion
- Refund and chargeback handling
- Transaction logging and audit trails
- PCI compliance support and security
- Webhook handling for payment confirmations

Architecture Example (based on real puqStripe implementation):

```

<?php

use App\Modules\Payment;

class StripePaymentGateway extends Payment
{
    public function getModuleData(array $data = []): array
    {
        return [
            'publishable_key' => $data['publishable_key'] ?? '',
            'secret_key' => $data['secret_key'] ?? '',
            'webhook_secret' => $data['webhook_secret'] ?? '',
            'sandbox' => $data['sandbox'] ?? false,
        ];
    }

    public function getClientAreaHtml(array $data = []): string
    {
        $invoice = $data['invoice'];
        $amount = $invoice->getDueAmountAttribute();
        $currency = $invoice->client->currency->code;

        $stripe = new StripeClient($this->module_data);

        $session = $stripe->createSession(
            referenceId: $invoice->uuid,
            invoiceId: $invoice->number,
            description: 'Invoice # . $invoice->number,
            amount: $amount,
            currency: $currency,
            return_url: $this->getReturnUrl(),
            cancel_url: $this->getCancelUrl(),
        );
    }
}

```

```

    );

    return $this->view('client_area', ['session' => $session]);
}

public function getSettingsPage(array $data = []): string
{
    $data['webhook_url'] = route('static.module.post', [
        'type' => 'Payment',
        'name' => 'puqStripe',
        'method' => 'apiWebhookPost',
        'uuid' => $this->payment_gateway_uuid
    ]);

    return $this->view('configuration', $data);
}
}

```







4. Notification Modules (Communication Channels)

Purpose: Send notifications through various communication channels

Real-World Examples:

- **puqSMTP:** Professional SMTP email delivery
- **puqPHPmail:** Basic PHP mail functionality
- **puqBell:** Push notification system
- **SMS Gateways:** Twilio, Nexmo, local SMS providers
- **Slack/Discord:** Team communication integration

Key Capabilities:

-  **Multi-channel message delivery**
-  **Template management and personalization**
-  **Delivery tracking and status reporting**
-  **Retry mechanisms for failed deliveries**
-  **Rate limiting and quota management**
-  **Rich content support (HTML, attachments, etc.)**

Architecture Example (based on real puqSMTP implementation):

```
<?php
```

```

use App\Modules\Notification;
use Illuminate\Support\Facades\Mail;

class SMTPNotificationService extends Notification
{
    public function getModuleData(array $data = []): array
    {
        return [
            'email' => $data['email'] ?? '',
            'server' => $data['server'] ?? '',
            'sender_name' => $data['sender_name'] ?? '',
            'port' => $data['port'] ?? 587,
            'encryption' => $data['encryption'] ?? 'tls',
            'username' => $data['username'] ?? '',
            'password' => $data['password'] ?? '',
        ];
    }

    public function send(array $data = []): array
    {
        try {
            $validator = Validator::make($data, [
                'to' => 'required|email',
                'subject' => 'required|string|max:255',
                'message' => 'required|string',
            ]);

            if ($validator->fails()) {
                return [
                    'status' => 'error',
                    'message' => $validator->errors(),
                    'code' => 422,
                ];
            }

            $mailConfig = $this->buildMailerConfiguration();

            Mail::mailer($mailConfig)->send(new NotificationMail(
                $data['to'],

```

```

        $data[' subject' ],
        $data[' message' ],
        $data[' attachments' ] ?? []
    ));

    $this->logInfo(' send' , ' Email sent successfully' , [
        ' to' => $data[' to' ],
        ' subject' => $data[' subject' ]
    ]);

    return [' status' => ' success' ];

} catch (Exception $e) {
    $this->logError(' send' , ' Email sending failed' , $e->getMessage());
    return [' status' => ' error' , ' message' => ' Failed to send email' ];
}
}
}

```

How Modules Integrate with PUQcloud

Lifecycle Management

Every module follows a standardized lifecycle:

1. **Installation** Module files are uploaded to the correct directory structure
2. **Discovery** System automatically detects and registers the module
3. **Activation**: `activate()` method creates databases, configures settings
4. **Operation** Module handles requests and performs designated functions
5. **Updates**: `update()` method handles version migrations and schema changes
6. **Deactivation**: `deactivate()` method cleans up resources and data

Integration Points

Admin Interface Integration

```

// Navigation menu configuration
public function adminSidebar(): array

```

```

{
    return [
        [
            'title' => 'Server Management',
            'link' => 'servers',
            'active_links' => ['servers', 'server-groups'],
            'permission' => 'manage-servers'
        ]
    ];
}

// Route definitions with permissions
public function adminWebRoutes(): array
{
    return [
        [
            'method' => 'get',
            'uri' => 'servers',
            'permission' => 'view-servers',
            'name' => 'servers',
            'controller' => 'ServerController@index'
        ]
    ];
}

```

Client Area Integration

```

// Multi-tab client interface
public function getClientAreaMenuConfig(): array
{
    return [
        'overview' => [
            'name' => 'Service Overview',
            'template' => 'client_area.overview'
        ],
        'management' => [
            'name' => 'Account Management',
            'template' => 'client_area.management'
        ],
        'statistics' => [

```

```
        'name' => 'Usage Statistics',  
        'template' => 'client_area.statistics'  
    ]  
];  
}
```

Revision #5

Created 30 July 2025 02:25:18 by Dmytro Kravchenko

Updated 13 August 2025 05:11:33 by Dmytro Kravchenko